

École polytechnique de Louvain

Comparative study of multi-variate metaheuristic algorithms: a case study of the hydrogen supply chain design including production and transport technologies in Belgium

Author: **Arthur GENGLER**

Supervisors: **Francesco CONTINO, Davide TONELLI**

Reader: **Geovani GRAPIGLIA**

Academic year 2021–2022

Master [120] in Electrical Engineering

Abstract

In the last two decades, metaheuristic algorithms have progressively emerged as promising methods to identify near-optimal solutions of complex optimization problems. Beyond their initial development specific to the solution of NP-hard optimization problems, recent versions of these algorithms have been proposed for the solution of Mixed Integer Linear (MILP) and Nonlinear (MINLP) optimization problems. This is the case for the class of Multi-Variate (MV) metaheuristics, capable of handling both continuous and discrete decision variables for application to engineering optimization problems.

The present work focuses on the identification of the best Multi-Variate metaheuristic algorithm for the solution of the complex problem of the nonlinear hydrogen supply chain design (HSCD), generally studied in its mixed-integer formulation and solved with mathematical programming. Five Multi-Variate metaheuristics are considered: Ant Colony Optimization, Particle Swarm Optimization, Genetic Algorithms, Differential Evolution and Adaptive Estimation of Distribution Algorithms. First, this master thesis explains the structure of each algorithm, then it studies the hyperparameters specific to each metaheuristic which maximize their performance. By comparing the five algorithms on the same problem instance, the best is selected and applied to the optimization of the HSCD in Belgium.

All these algorithms are iterative algorithms based on the use of the same solution archive which stores tentative solutions in a matrix of dimension $(k, nR + nO)$, where k is the number of tentative solutions temporarily stored in the memory and $nR+nO$ is the total number of decision variables handled by the algorithm. The key difference among these algorithms consists in the construction of new tentative solutions. In the multi-variate version of Ant Colony Optimization (ACO) a new solution is computed based on the variance and the mean of the solution archive. In Particle Swarm Optimization (PSO) the vectors in the solution archive are considered as the position of a moving particles and their positions change between two iterations according their respective velocity. In Genetic Algorithm (GA) a crossover operator uses the information of two parents to create two new solutions, then the mutation operator creates a perturbation around those new solutions. Differential Evolution (DE) creates a mutant solution composed of a tentative solution and the difference between two other tentative solutions. The crossover operator exchanges part of this mutant with part of yet another tentative solution to build the new solution. Adaptive Estimation of Distribution Algorithm (AEDA) constructs a distribution, whose mean and variance are calculated based on part of the solution archive, the new solutions are then created by sampling this distribution. Adaptive Estimation of Distribution (AEDA) uses both a Gaussian and a Cauchy distribution. For all these algorithms, the discrete variables of the Multi-Variate version are handled by mapping the discrete search space to the continuous domain.

As part of this work, for Genetic Algorithm, the HX-PM (Heuristic Crossover-Power Mutation) is found to be the best type of Genetic Algorithm for application with the solution archive and for dealing with discrete variables. A sensitivity analysis is performed in order to tune these metaheuristic algorithms in order for them to reach their maximal potential. It is showed that the performances of each metaheuristic are highly dependent on the choice of their hyperparameters. In addition, a combination of hyperparameter values, which have showed good performances on a benchmark function, is proposed.

Finally, this work identifies the Adaptive Estimation of Distribution as the best choice (among the five metaheuristic algorithms considered) to design the hydrogen supply chain in Belgium as MINLP optimization problem. The Ant Colony Optimization is also a good choice for this specific optimization problem as it finds solutions close to the best one found by AEDA in less iterations. Regarding Differential Evolution and Particle Swarm Optimization they both have shown good results, but take more time to converge. Also, the specific HX-PM Genetic Algorithm considered in this study is not recommended to design the hydrogen supply chain in Belgium. Finally, the reduction in computational time obtained by the introduction of a discretization of real decision variables is considered not worth for the HSCD, due to the associated high reduction in the accuracy of the results.

Acknowledgments

I would like to express my gratitude to my supervisor Prof. Francesco CONTINO and to Davide TONELLI for proposing me this interesting master thesis and for having accepted me in the team of students they supervise. Moreover, they were always nice and full of proposition for the realisation of this work.

I would particularly like to thank Davide TONELLI for taking the time, every week, to meet with me in order to give me advice and answer the different questions I had as well as for helping me in the redaction of this master thesis. His contribution is essential to the accomplishment of this master thesis.

Finally, I would like to thank the team directed by Prof. Francesco CONTINO and Prof. Hervé JEAN-MART for allowing me to present my advancements during a mid-term presentation and for giving me useful feedbacks.

Contents

1 Introduction	6
1.1 Context and motivations	6
1.2 Master thesis objectives	7
1.3 Master thesis outline	7
2 Theory of Metaheuristics	8
2.1 Basic metaheuristic structure	9
2.1.1 Initialization	9
2.1.2 Selection of solutions	11
2.1.3 Construction of new solutions	14
2.1.4 Update of the population	15
2.2 Termination criterion	15
2.3 Genetic Algorithm	17
2.3.1 Laplace crossover	18
2.3.2 Heuristic crossover	20
2.3.3 Non-uniform mutation	20
2.3.4 Power mutation	21
2.3.5 Summary GA	22
2.4 Differential Evolution	23
2.4.1 Differential mutation	23
2.4.2 Differential crossover	23
2.4.3 Summary DE implementation	24
2.5 Particle Swarm Optimization	26
2.5.1 Summary PSO implementation	29
2.6 Ant Colony Optimization	31
2.6.1 Summary ACO	32
2.7 Adaptive Estimation of Distribution Algorithm	33
2.7.1 Summary AEDA	34
3 Benchmark functions	35
4 Real case problem	38
4.1 Objective function	38
4.2 Constraints	39
4.3 Implementation	41
5 Methodology	42
6 Results	45
6.1 Choice of algorithm for GA	45
6.2 Choice of Algorithm for DE	47
6.3 Sensitivity Analysis of hyperparameters specific to a metaheuristic	49
6.3.1 ACO	49
6.3.2 AEDA	51
6.3.3 PSO	53
6.3.4 GA	54
6.3.5 DE	59
6.3.6 Sensitivity Analysis of common hyperparameters	60
6.3.7 Comparison with tuned hyperparameters	64
6.3.8 Discussion and remarks	66
6.4 Results on Real case	68

6.4.1	Choice of algorithm	68
6.4.2	Impact of discretization	69
7	Conclusion	73
8	Perspectives	74

List of Figures

1	Illustration of diversification vs intensification, From [3]	8
2	Flowchart representing the steps of a metaheuristic algorithm	9
3	Representation of a solution vector	11
4	Solution archive	11
5	Distribution of weight with $q=0.03$, $q=0.05$, $q=0.08$ and $q=0.12$	12
6	Empiric distribution with with M equal 1, 3, 5 and 10	13
7	Probability of choosing a solution with lots of/few variations in the fitness	13
8	Construction method for new solutions	15
9	One point, two point and uniform crossover	18
11	Evolution of the behavior of non-uniform mutation with time for $x^t=10$, $x_u=15$, $T=1000$, $u=0.5$, $r=0.6$	20
12	Power distribution for $p=0.1$, $p=0.25$, $p=0.5$ and $p=1$	21
13	All, Ring, Four clusters, Pyramid and Square topology, From [29]	27
14	Cauchy vs Gaussian distribution	28
15	Representation of the population and the associated variables. Xr_j^i is a real decision variable, Xo_j^i it a discrete decision variable, Vr_j^i is the velocity variable associated to Xr_j^i , Vo_j^i is the velocity variable associated to Xo_j^i , $f(X_j)$ is the fitness of the particle at rank j based on its position vector, w_j and S_j are variables associated to the particle at rank j	29
16	Distribution for discrete and continuous variable, from [32]	31
17	Benchmark functions in 3D	36
21	Curve of convergence of GA depending on crossover type on benchmark functions	45
22	Boxplot for different crossover type on benchmark functions over 5 runs	46
23	Curve of convergence of GA depending on mutation type on benchmark functions	46
24	Boxplot for different mutation type on benchmark functions over 5 runs; NUM= non-uniform mutation, PM = power mutation	47
25	Curve of convergence of DE depending on DE type on benchmark functions	47
26	Boxplot for different DE type on benchmark functions over 5 runs	48
27	Curve of convergence of DE depending on cross type on benchmark functions	48
28	Boxplot for different DE crossover type on benchmark functions over 5 runs	49
29	Curve of convergence of ACO depending on xi on benchmark functions	49
30	Boxplot for different xi type on benchmark functions over 5 runs	50
31	Curve of convergence of ACO depending on xi on benchmark functions	50
32	Boxplot for different q on benchmark functions over 5 runs	51
33	Curve of convergence of AEDA depending on NP_{best} on benchmark functions	51
34	Boxplot for different NP_{best} on benchmark functions over 5 runs	52
35	Curve of convergence of PSO depending on c on benchmark functions	53
36	Boxplot for different c on benchmark functions over 5 runs	53
37	Curve of convergence of GA depending on cpx type on benchmark functions	54
38	Boxplot for different cpx on benchmark functions over 5 runs	54
39	Curve of convergence of GA depending on mp type on benchmark functions	55
40	Boxplot for different mp on benchmark functions over 5 runs	55
41	Curve of convergence of GA depending on StrengthPM type on benchmark functions	56
42	Boxplot for different strength for power mutation on benchmark functions over 5 runs	56

43	Curve of convergence of GA depending on tr_size type on benchmark functions	57
44	Boxplot for different tournament size on benchmark functions over 5 runs	57
45	Curve of convergence of GA depending on elite_size type on benchmark functions	58
46	Boxplot for different elite size on benchmark functions over 5 runs	58
47	Curve of convergence of DE depending on F type on benchmark functions	59
48	Boxplot for different F on benchmark functions over 5 runs	59
49	Curve of convergence of DE depending on CR type on benchmark functions	60
50	Boxplot, for different CR on benchmark functions over 5 runs	60
51	Impact of m on the early stage of the curve of convergence for ACO/AEDA/PSO/DE	61
52	Impact of k on convergence for the five metaheuristics	63
53	Comparison of the five metaheuristics on all the benchmark functions	65
54	Boxplot for different metaheuristic on benchmark functions over 5 runs	66
55	Curve of convergence of ACO, AEDA, DE, PSO and GA on the HSCD	68
57	Curve of convergence depending on the discretization level with AEDA	69
59	Boxplot of the number of iterations before stagnation depending on the discretization level with AEDA	70
60	Curve of convergence depending on the discretization level with ACO	71
62	Boxplot of the number of iterations before stagnation depending on the discretization level with ACO	72

1 Introduction

This section explains the context and motivations which led to the realisation of the master thesis as well as its four main goals and the outline.

1.1 Context and motivations

One of the major challenges of our time is global warming, which is strongly correlated to the increase of CO₂ emissions in the atmosphere. To face this challenge, hydrogen is expected, amongst other solutions, to play a significant role in a future low-carbon energy landscape, covering up to 18% of the global energy demand by 2050 according to the Hydrogen Council [1]. However, in order to allow a higher penetration of hydrogen in the energy system, the development of its infrastructure is necessary on the production, the transport and the demand side. In this context, studying the optimal design of the hydrogen supply chain is required in order to build a cost-efficient hydrogen-based economy.

In the literature, this analysis has often been carried out as an optimization problem with different mathematical formulations, depending on the approximations introduced. The most widespread formulation is based on Mixed-Integer Linear Programming (MILP). Despite the fact that MILP solvers are very efficient, their formulation still requires a linearization of any possible non-linearities. Generally, this is possible by introducing additional auxiliary binary variables which allow, e.g., a piece-wise linearization of the nonlinear functions. However, some nonlinearities (like the pressure drops in a pipeline system) are not equally linearizable. In such cases, the problem has to be formulated as a nonlinear (NLP) or mixed-integer nonlinear (MINLP) optimization problem.

The existing methods to solve MINLP problems can be classified in two groups [2]: (i) based on a single design point in the decision variables space which incrementally moves towards a minimum and (ii) population methods based on a collection of design points. The first group includes descent direction methods (e.g., Newton method) based on the information of the gradient of the objective function, or stochastic methods. The second group includes metaheuristic algorithms, which also rely on stochasticity.

This master thesis focuses on metaheuristics algorithms as they allow to find good quality solution in a reasonable execution time even for nonlinear optimization problems. Furthermore, their stochastic behaviour as well as their high-level robust search strategies enhance the possibility of escaping from local optima. These advantages have led to a wide use of metaheuristic algorithms in the last 25 years for solving real-world optimization problems and, when taking into account the precision of the solution and the computational time of the methods, metaheuristics algorithms have often showed the best performances [3].

A characteristic of metaheuristic algorithms is that, from a very general point of view, no metaheuristic can be considered a priori better than another. According to the No Free Lunch (NFL) theorem [4] any two metaheuristic algorithms are equivalent when their performance is averaged across all possible problems. Furthermore, because metaheuristic algorithms don't rely on strong mathematical property, it is impossible to know in advance which one will perform better on the specific problem of the hydrogen supply chain design (HSCD). Hence, a comparison of state of the art metaheuristic algorithms on this specific optimization problem is required.

The algorithms selected for the present comparison have been chosen from the paper [5] as they can solve MINLP problems by encoding both continuous and discrete variables (the latter including also binary variables): Multi-Variate Ant Colony Optimization (ACO-MV) [6], Multi-Variate Particle Swarm Optimization (PSO-MV) [5], Multi-Variate Differential Evolution (DE-MV) [7], Multi-Variate Adaptive Estimation of Distribution Algorithm (AEDA-MV) [8]. A specific Genetic Algorithm allowing to deal with both continuous and discrete variables has been added to the comparison (GA-MV) [9].

1.2 Master thesis objectives

The first objective of this master thesis is to identify if it is possible to use a Genetic Algorithm to solve MINLP problems while using the same encoding scheme of ACO-MV, PSO-MV, DE-MV and AEDA-MV, already existing in the literature.

Secondly, this work reviews the aforementioned state of the art population-based metaheuristics capable of dealing with both integer and continuous variables. The structure of this review aims at underlying the characteristics shared by the the metaheuristics and the key differences based on which they differ. Concerning the algorithms, this thesis focuses on Genetic algorithm (GA), Differential Evolution (DE), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Adaptive Estimation of Distribution Algorithm (AEDA).

Thirdly, each of these metaheuristics have specific hyperparameters which highly affect the performance of the algorithm. It is admitted that it is not clear what the best settings are nor how to tune these hyperparameters [10]. Furthermore, most of the tuning performed in the literature is opaque and doesn't explain the choices made for those hyperparameters. In this master thesis we propose a simple but systematic way of tuning the hyperparameters based on a sensitivity analysis. We justify the choices made at each step and present the impact of each hyperparameter on the final results.

Finally, we compare the tuned metaheuristic algorithms on the optimization problem of the hydrogen supply chain design, and we select the best performing algorithms. A study of the impact of discretization on the best performing algorithm is then performed. It allows to see if there exist a level of discretization such that it converges faster, than with real variables, while keeping a good accuracy on the final result. The algorithms are applied to the case study of Belgium for the decarbonization of the industrial demand.

1.3 Master thesis outline

This report is structured as follows. Section 2 introduces to the general theory around metaheuristics followed by a detailed explanation of the specificity of the five metaheuristics (Genetic Algorithm in 2.3, Differential Evolution in 2.4, Particle Swarm Optimization in 2.5, Ant colony Optimization in 2.6 and Adaptive Estimation of Distribution Algorithm in 2.7). Section 3 explains the benchmark functions used for the sensitivity analysis as well as their role and what are their limitations. Section 4 describes the formulation of the HSCD accompanied by the structure of its implementation. Section 5 presents the methodology for tuning the hyperparameters as well as for identifying the best metaheuristic algorithm. Section 6 identifies the specific GA and DE algorithms implemented in this work and studies their performance on benchmark functions. The results of the sensitivity analysis used to tune the five algorithms are presented. Then, the performance of the tuned metaheuristics is analysed on the case study of HSCD. Section 7 is the conclusion with the main findings. Finally, some perspectives for future works are proposed in section 8.

2 Theory of Metaheuristics

This section introduces the basic principle of metaheuristics as well as the structure of their implementation and different steps performed by a metaheuristic algorithm. Then a detailed explanation of the theory about the five metaheuristics studied is given.

Originally, metaheuristics were designed as a high-level problem solver. The term "meta" refers to the high level strategy of metaheuristics algorithms which use a set of heuristic techniques in order to guide the search process. While a "heuristic" algorithm is design to solve a specific problem, "metaheuristics" have been conceived as problem-independent. Most of the time, a deterministic optimization technique is designed to solve a particular problem. For instance the conjugate gradient method works well for quadratic optimization problems, but it is not appropriate for solving multimodal problems[3].

The search strategy of metaheuristics is based the exploration and the exploitation of the solutions in the population. The exploitation, also called intensification, is responsible for searching around the best solution found so far, while the exploration, also called diversification, is responsible for discovering new potential optima far from the search area already evaluated. These two concepts are illustrated on the Figure 1, where the green area represents the search space in the case of diversification and intensification.

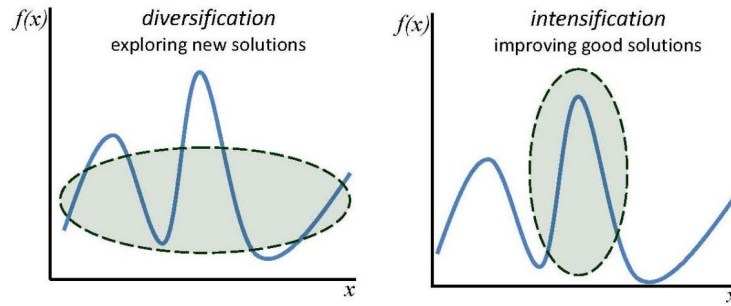


Figure 1: Illustration of diversification vs intensification, From [3]

In all metaheuristics there is a trade-off between exploration and exploitation. If we prioritize too much exploitation over exploration, the search of the algorithm for new potential solutions will be focused close to the highest quality solution already discovered. Hence, if this high quality solution corresponds to a local minimum, the algorithm will never find the global minimum. On the other hand, if we prioritize too much exploration over exploitation the search of the algorithm is too scattered. This behaviour is similar to a random search and might therefore take a long time to find the global optimum.

The focus of this master thesis is on the five following metaheuristics: Genetic algorithm (GA), Differential Evolution (DE), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Adaptive Estimation of Distribution Algorithm (AEDA).

These five selected algorithms have the following common characteristics:

- population-based: the algorithm is based on a population of tentative solutions, stored in the solution archive, of the optimization problem,
- iterative evolution of the population: the modification of the solution archive is based on operators (e.g., mutation, crossover, ...) subject to at certain level of randomization,
- survival of the fittest: fitter solutions are allowed to pass onto the next generation more frequently,

- stochasticity: the achievement of a global optimum is never guaranteed.

In all of these five metaheuristics there is no use of derivative. This allows these algorithms to be applicable to a lot of problems, but it also implies the need of a strategy to guide the search process. Metaheuristics evaluate a solution by computing its fitness value, which is based on the function evaluation. A classical optimization problem is the Travelling Salesman Problem (TSP) where the algorithm has to find the shortest path that a salesman should take to go through a list of cities and return to the origin city. In the case of TSP the function evaluation computes the total distance of a route and a smaller distance corresponds to a better fitness value. Often in real-world problems the function evaluations is the most time-consuming part of the metaheuristic algorithms.

The five considered metaheuristics have different inspirations. The Genetic Algorithm as well as the Differential Evolution find their inspiration in the evolution and more specifically on the reproduction between two parents. The Ant Colony Optimization and the Particle Swarm Optimization are inspired from the swarm intelligence, which regroups techniques based on the cooperative behavior of agents (ants or particle). The Adaptive Estimation of Density Algorithm it is based on statistics.

2.1 Basic metaheuristic structure

To understand how metaheuristics work, here under a basic flowchart presents the different steps detailed in the following subsections.

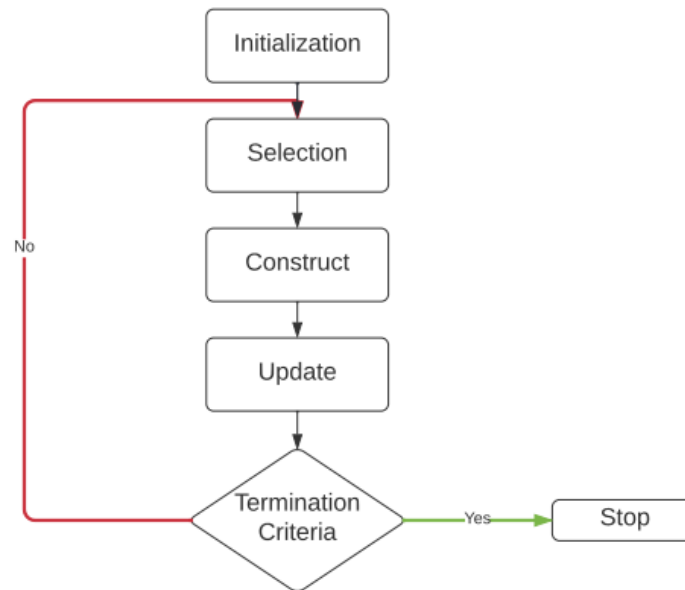


Figure 2: Flowchart representing the steps of a metaheuristic algorithm

2.1.1 Initialization

The first step of the algorithm is to initialize the population. To design the population we have to take into consideration 3 elements: the way a solution is encoded, the size of the population and the way the initial population is generated.

The three main techniques to encode the population are Binary, Gray and Real coding.

Binary encoding. Binary encoding is based on coding an integer parameter directly in its base-2 representation, using bits, and then applying a standard binary-to-integer mapping to decode the parameter value. A problem of this representation is that, while performing mutation and crossover operations we may change the value of a single bit which can lead to a new real value which is really different from the previous one. It also faces the “Hamming cliff” problem. An example of the Hamming cliff problem is that for the transition from 7 (0111 in binary coding) to 8 (1000 in binary) all the bits need to change. One way to avoid this Hamming cliff problem is to use the Gray coding. [11]

Gray encoding. The reflected binary code (RBC), also known just as reflected binary (RB) or Gray code, is an ordering of the binary numerical system such that two successive values differ in only one bit. For example, the representation of the decimal value "1" in binary would normally be "001" and "2" would be "010". In Gray code, these values are represented as "001" and "011". That way, incrementing a value from 1 to 2 requires to change only one bit instead of two. In the Gray code, the Hamming distance is always one for any two solutions value that are differing by one. That is not the case in the standard binary code where a single bit flip (a bit go from 0 to 1 or 1 to 0) at the most significant position dramatically changes the value. There are many existing different gray codes: n-ary Gray code, Balanced Gray code, Long run Gray codes, Monotonic Gray codes, Beckett–Gray code, Snake-in-the-box codes, Single-track Gray code and Two-dimensional Gray code. [12]

This description of Binary and Gray encoding focuses on integer numbers but those techniques can be extended to the real world with a certain precision. The more bits are used to approximate a real number, the more precise it is. A major drawback of Binary/Gray encoding is that they face difficulties when applied to problems having large search space and seeking high precision. Indeed, in Binary/Gray encoding, the length of the solution vector must be chosen to get a desired degree of precision in the final solution. Moreover, some additional computational time is required to go from Binary/Gray encoding to real value, to finally compute the fitness of a solution. To overcome these difficulties real-encoding are often used nowadays. This is also the choice we will make for this thesis.

Real encoding. Real encoding is a quite natural way of representing a solution, where each variable directly takes its real value represented in a base-10 and allows floating point numbers.

The continuous variables will therefore be encoded using the real representation. To be able to work also with discrete variables the continuous relaxation approach proposed by [6] is used. It allows to use the same operators for discrete variables the ones for continuous variables. The metaheuristic algorithms do not operate on the actual values of the discrete variables, but instead on their indices in an array. For example, if a discrete variable can take the values [1.5, 5, 9.2] we associate to each variables an index {0:1.5, 1:5, 2:9.2}. The algorithm will work on the relaxed indices, meaning a continuous domain with extremes 0 and 2 in the previous example. However, before evaluating the fitness value, the continuous values of the discrete variables are rounded to the nearest integer, and the value at that index is used for the computation of the solution fitness. This simple method also allows to deal with non-numerical ordered variables such as [small, medium, large] by working with the indices {0:small, 1:medium, 2:large}

Here under is the representation of a solution vector where R_j^i is the real decision variables and O_j^i are discrete decision variables, where i is the index of the decision variable in the solution vector, and j is the index of the solution vector in the solution archive. Both decision variables are encoded with the real encoding method.

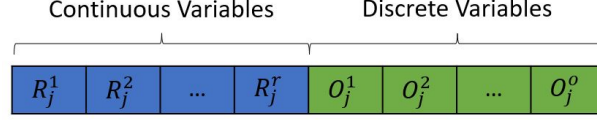


Figure 3: Representation of a solution vector

As we are working with population-based algorithms we initialize a certain number, k , of solution vectors. The hyperparameter k has a direct impact on the convergence rate. If the size of the population is too small it increases the probability to converge towards a local minimum. We lack of diversity in the population and are unable to explore the search space appropriately. But, since the computational time increases with the size of the population, we have to find a compromise. Unfortunately, there is no predefined size and it is a parameter that changes according to the dimension of the problem. The complete set of k solution vectors is represented as a matrix and is called solution archive. The position in the solution archive is based on the fitness value, $f(X_i)$ of the solution vector X_i , X_1 being highest quality solution vector.

	Continuous Variables			Discrete Variables			
X_1	R_1^1	...	R_1^r	O_1^1	...	O_1^o	$f(X_1)$
X_2	R_2^1	...	R_2^r	O_2^1	...	O_2^o	$f(X_2)$
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
X_j	R_j^1	...	R_j^r	O_j^1	...	O_j^o	$f(X_j)$
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
X_k	R_k^1	...	R_k^r	O_k^1	...	O_k^o	$f(X_k)$

Figure 4: Solution archive

We create the initial population by sampling randomly within the range of lower and upper bounds of the variables domain. In some cases, based on the knowledge that we have of the problem, this step can be improved using heuristic methods to find a better initial population to converge faster to the optimal solution.

When using a rank selection (see section 2.1.2), the weight associated to each solution vector is computed during the initialization step.

2.1.2 Selection of solutions

The selection process is used to choose current solutions (also called parents) that will be used as base for the construction of new solutions (also called offsprings). There are many ways of selecting parents amongst which the most common are: wheel of fortune selection, rank selection, tournament selection, all population selection and NP-best selection.

Wheel of fortune selection. The probability of a solution to be selected is proportional to its fitness score [13].

$$\text{Prob}_j = \frac{f_j}{\sum_{i=1}^k (f_i)} \quad (1)$$

where Prob_i is the probability of selecting solution i , f_i is the fitness value of solution i and k is the size of the population.

Selection by rank. The solutions are ranked based on their fitness and the lower the rank is, the better the fitness score is. A weight is associated to each solution according to its rank, and is computed with the following equation [6]:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(\text{rank}(j)-1)^2}{2q^2k^2}}, \quad (2)$$

where k is the size of the population, $\text{rank}(j)$ is the rank of the solution j and q is an hyperparameter that influences the probability distribution and thus the selection pressure towards high quality solutions.

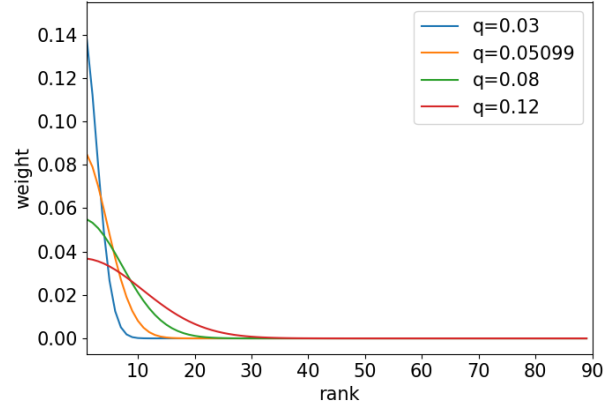


Figure 5: Distribution of weight with $q=0.03$, $q=0.05$, $q=0.08$ and $q=0.12$

In Figure 5 we see that higher values of q lead to a more uniform selection.

The probability of selecting an element j in the population is computed as follows:

$$\text{Prob}_j = \frac{\omega_j}{\sum_{i=1}^k \omega_i}, \quad (3)$$

where w_i is the weight of element i and is computed according to equation 2.

Uniform selection. The probabilities of each solution to be selected are equal.

Selection by tournament. This technique uses random selection of M solutions, M is called the tournament size. The, the best one is selected based on the competition among these M solutions. If M is equal to the size of the population, it is equivalent to selecting the best solution and if $M=1$ this is equivalent to a random selection. This hyperparameter allows to adapt the selection pressure. The selection pressure is used as in the biological world, a high selection pressure will encourage fitter solutions to be selected. To illustrate this variation of selection pressure, the following figure represents the frequency a solution has been chosen. To have a good representation of the distribution, the tournament selection was performed 10^6 times.

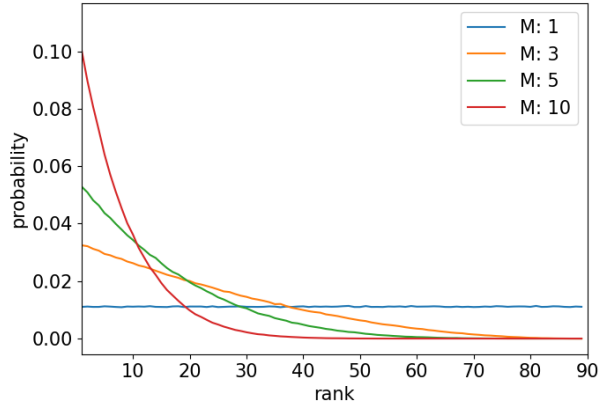


Figure 6: Empiric distribution with with M equal 1, 3, 5 and 10

All population selection. Here we simply select all the population as a base to construct the new population.

NP-best selection. The hyperparameter NP-best determines how many solutions are used to build the new population.

Wheel of fortune and rank selection are quite similar, but it's important to notice some specificities. First, the wheel of fortune selection does not require to sort the population. This is beneficial in terms of computational time. Despite this advantage, rank selection is often preferred because it works with negative fitness values and allows to keep the selection pressure at every iteration of the algorithm. Indeed, if every solution has very close fitness scores in the wheel of fortune (which happens when the algorithm converges), each solution has the same probability to be selected. To illustrate this phenomenon here is a graph representing the probability of choosing a solution among 90 in two cases. In the first case, the fitness of each solution varies from 10 to 100 with a constant step and in the second case the fitness varies from 10 to 11 with a smaller constant step.

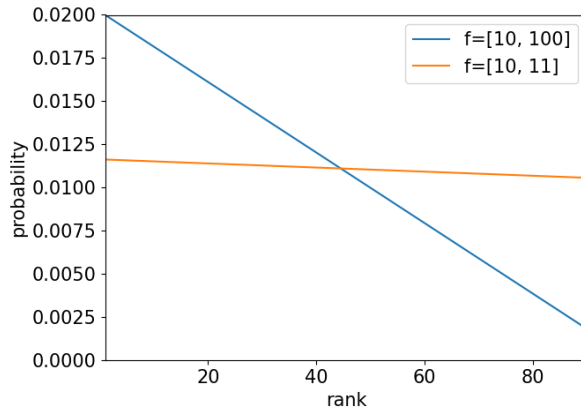


Figure 7: Probability of choosing a solution with lots of/few variations in the fitness

This characteristic is often not desired as we wish to keep a selection pressure during all the iterations,

which is not the case when using a wheel of fortune selection. Indeed, in Figure 7 we see the probability of choosing the best solution (rank=1) is similar to choosing the worst solution (rank=90) when the variation in terms of fitness in the population is small.

In terms of diversity achieved by the type of selection we have a maximum of diversity when selecting all the population. For the rank, NP-best and tournament selection we can choose a certain level of diversity by playing with the parameters q , NP-best and M . It is interesting to note that for the selection there is a trade-off between diversity and selection pressure which is equivalent to a trade-off between exploration and exploitation.

2.1.3 Construction of new solutions

The construction of new solutions is done by what is called an operator. An operator is for example the crossover operation in genetic algorithms. Each metaheuristic has its own operators and those concerning GA, DE, PSO, DE, AEDA are detailed in the following section.

There are three possible strategies to build a solution in the case of MVOPs (Mixed-Variable Optimization Problems). The first one is to use different operators to handle continuous and discrete variables separately. This requires to double the number of operators for each algorithm in order to be able to deal with both discrete and continuous variables. This approach often leads to a large number of objective function evaluations [6]. The second method is the discretization method. Here the continuous variables are discretized and the operators work only with discrete variables. In this case, poor performance is expected when applying the method to a problem with a high degree of precision. The third one is the relaxation method. Here the discrete variables are relaxed and considered as continuous variables by the different operators. The next step is to go back in the discrete world for those variables in order to evaluate the objective function. To go back to the discrete world truncation and rounding are generally used. We know that, with a classical continuous solver, relaxation does not always find the optimum solution, but here, because we find the optimum by iteration, we can hope that most of the time the algorithm will nevertheless reach the global minimum. This method also allows to limit the number of operators and facilitate the implementation. The relaxation method is chosen despite the fact that its performance may decrease when the number of discrete variables increases.

The algorithm will however consider the continuous and discrete variables separately, as showed in the figure below, because the discrete decision variables represent indices (expressed as real values) in the solution archive.

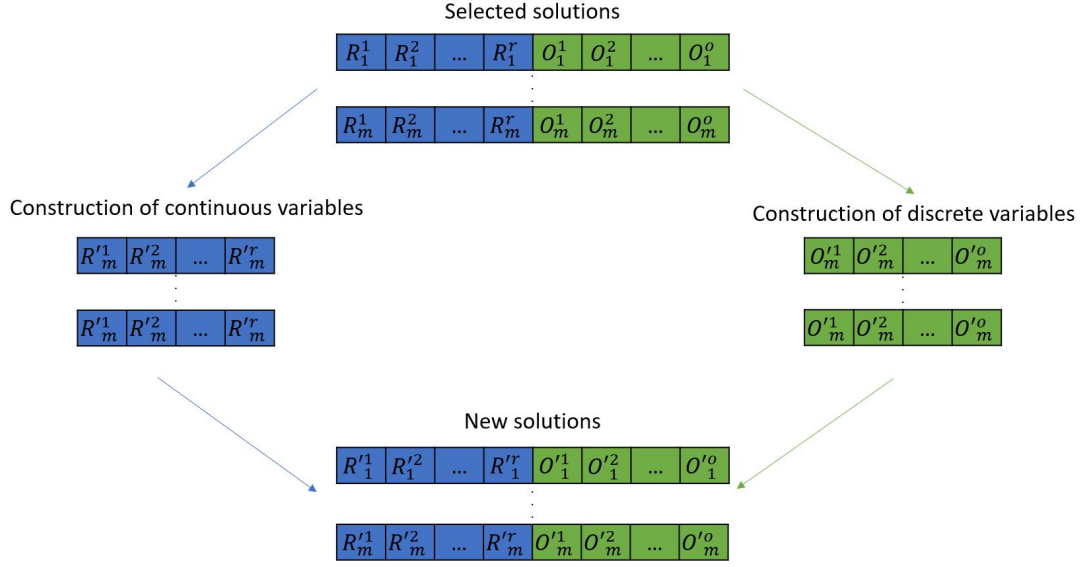


Figure 8: Construction method for new solutions

2.1.4 Update of the population

The update the population, with the new constructed, keeps the population size constant. There are many ways of updating the population amongst which the most common are: Delete-all, Elitism, Weak elitism, Replacement and k-best.

Delete-all. The old population is deleted and replaced by the new solutions. This is probably the most common technique and is widely used due to its relative ease of implementation. It is also free of hyper-parameters unlike the other techniques. However, there is still a risk of losing the best solution found so far [14].

Elitism. It is similar to the delete-all technique. However, with elitism, the n best solutions vector of the solution archive are kept. Hence, the best solutions found so far is never lost. This technique is often used in Genetic algorithms and results in increasing the efficiency of such algorithms [14].

Weak elitism. It is a variant of the classical elitism. The new updated solution archive is the k best solutions among $n+m$ solutions, where n is the elite size and m is the new created solutions. Thus, we keep the k best solutions among the m new created solution and a part of the old population.

Replacement. This method consist in replacing a solution vector in the solution archive by the new created solution only if the fitness value improves [5].

k-best. This method consist in keeping the k best solutions among $k+m$ solutions, where m is the number of new solutions. Thus we keep the k best solutions among the m new created solutions and the entire old population. [6]

2.2 Termination criterion

As our five metaheuristics are iterative algorithms, a termination criterion is needed. A first idea is to impose a certain precision. Unfortunately, as for many problems the optimum solution is unknown, this technique is not often applicable. The two most common termination criteria are, either a maximum number of iterations/function evaluations, or a maximum number of stagnation iterations/function evaluations. The number of stagnation iterations uses a parameter, ϵ , which is a threshold to determine if the improvement between two iterations is big enough or if it is considered as a stagnation. There are two ways to compute

this improvement. We can use either an absolute or a relative delta. The formulas are showed here under:

$$\Delta_{abs} = |f^{t-1} - f^t|, \quad (4)$$

$$\Delta_{rel} = \frac{|f^{t-1} - f^t|}{f^t}. \quad (5)$$

An absolute delta is more intuitive and allows to fix the desired level of precision in terms of floating point number, but it is problem specific. On the other hand, a relative delta is an indicator of percentage and is therefore not problem specific.

Using a restart strategy

The restart strategy was proposed in the paper [6] to fight stagnation. The way it works is as follows, after a certain number of iterations where there is no significant improvement, the algorithm restarts while keeping the best solution found so far. The idea is that, a stagnation can occur when there is not enough diversity in the solution archive. This particular case results in creating the same, or almost the same, solution over and over. Thus, introducing new values in the solution archive can help increase the diversity to find potentially better solutions.

Constraints handling

The metaheuristics do not use constraints to create new solutions. Hence, some of the solutions generated may be infeasible. To deal with constraints in the world of metaheuristics two methods were developed. Both methods are based on the idea that, a solution which respects all constraints should be preferred to one violating at least one. To do so, the metaheuristics play on the fitness values of the solutions.

The first method is called the death penalty method and rejects all infeasible solutions by assigning them a very large fitness value. The second method makes the difference among infeasible solutions by adding to the fitness a value which depends on the degree of constraint violation [5].

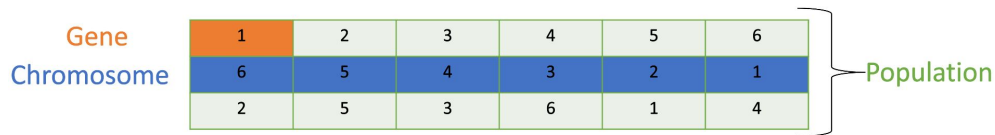
2.3 Genetic Algorithm

Genetic algorithms belong to the family of evolutionary algorithms. The inspiration for evolutionary algorithms comes from the observation that, the evolution of living beings tends to produce organisms which are more adapted to their environment. This is achieved by combining the genetic material of two parents to have an offspring, and by the Darwin's principle "Survival of the fittest" which exerts a natural pressure to keep the individual most suited to the environment.

The first mathematical foundation of GA was proposed by J.H. Holland [15] using the building block hypothesis. This hypothesis uses the notion of schema. A potential schema is, for example, 1**0*1 and describes the set of all words of length 6 with 1's at the first and sixth positions and a 0 at the fourth position. These highly fit schemata are called building blocks. The building block hypothesis supposes that, combining building blocks with a crossover operator will create highly fit solutions. In a way, by working with these building blocks, the complexity of the problem is reduced, since instead of building high-performance solutions by trying every conceivable combination, we construct better and better solution from the best partial solutions. This hypothesis also assumes that, the overall problem can be separated into smaller sub-problems.

Later, Goldberg [16] used this mathematical foundation to solve real optimization problems. The first GAs were used to solve discrete optimization problems using three main operators: selection, crossover and mutation. The selection operator is used to mimic the pressure selection and will select highly fit solutions. To create the offsprings, GAs use crossover and mutation. Crossover will perform mostly exploitation by using important information already present and passing it onto the child (offspring). The crossover operation allows pieces of well performing bit strings, called building blocks, to be exchanged by the parents to produce better children. Mutation performs exploration. Mutation is used to add diversity in the population and it helps to avoid premature convergence.

In GAs a solution vector is often called a chromosome and each variable of that chromosome is called a gene.



In the beginning, GAs were only used to solve discrete problems using crossover and mutation. The crossover operator generally operates on two parental chromosomes and exchanges information between those chromosomes in order to create new and possibly better offsprings [13]. The crossover operation allows building blocks to be exchanged [17]. The mutation operator, on the other hand, is responsible for a local and random perturbation in the offspring [13].

The first types of crossover were inspired by the biological crossover and they work in similar ways. Three classical crossover are describe here under.

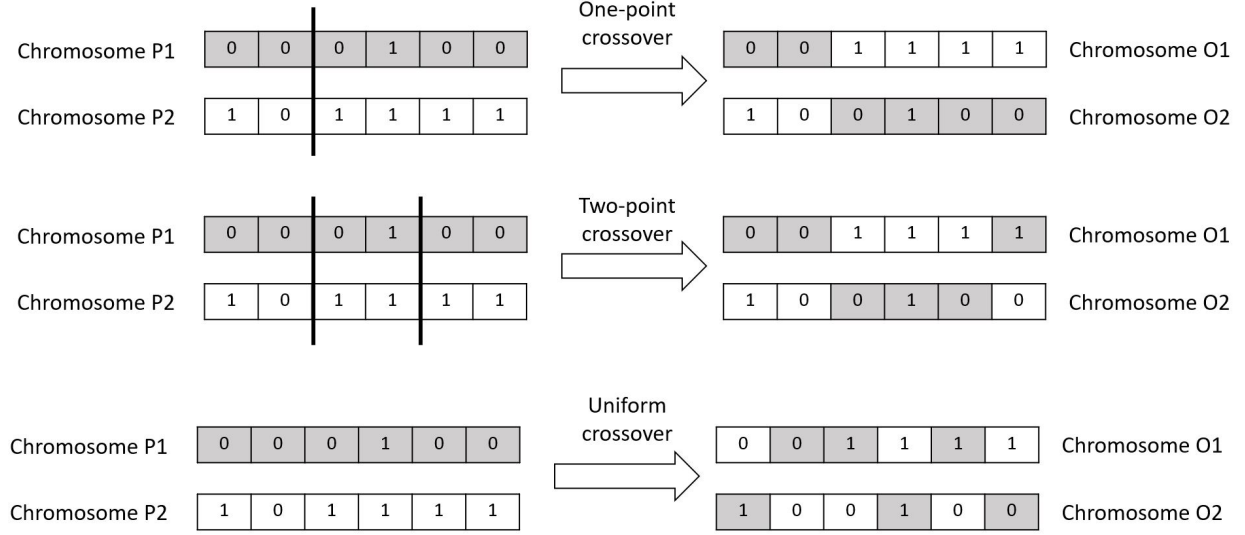


Figure 9: One point, two point and uniform crossover

The one point crossover randomly chooses a gene number beyond which the information presents in the chromosome P1 is exchanged with the information of the chromosome P2. The two point crossover randomly chooses two gene numbers. The exchange of information is carried out for the present gene between the two gene numbers. The uniform crossover goes through all the genes and for each of them randomly chooses to make, or not, an exchange of gene between the two chromosomes.

The classical types of mutations are the swap gene mutation, where two genes of a solution are swapped, and the bit-flip mutation where a certain number of genes are swapped from 0 to 1 or from 1 to 0. These operators can be used to solve combinatorial optimization problems using a binary encoding scheme. Later, new crossover and mutation operators (Ordered Crossover operator, Partially Mapped Crossover [18]) have allowed to work on combinatorial problem. They allow to use a real encoding scheme which have helped solving optimization problems like the TSP. For this particular problem, each gene represents a city and can take the value 1 to n , n being the number of cities. By iteratively performing crossover and mutation, we have a chance to obtain the optimal ordering. We already see in this case that, the bit-flip mutation isn't applicable for TSP due to the real encoding scheme. And when we switch to the continuous world, none of the above operators are efficient. Indeed, by only swapping the values of gene from one place to another we don't introduce new possible values. Thus, unless the initial population contains all the optimal values, which is unlikely in a real world problem for finite population size, the optimal solution will never be found.

To solve this issue, new crossover and mutation operators have been proposed. Two crossovers and two mutations which have showed good results on benchmark functions are the Laplace and the Heuristic crossover, the Non-uniform and the Power mutation [11] [19].

2.3.1 Laplace crossover

The density function of Laplace distribution is given by:

$$f(x) = \frac{1}{2b} e^{-\frac{|x-a|}{b}}, \quad (6)$$

where a is called the location parameter and b is a scale parameter. The distribution function of Laplace distribution is given by:

$$F(x) = \begin{cases} \frac{1}{2}e^{-\frac{|x-a|}{b}} & \text{if } x \leq a \\ 1 - \frac{1}{2}e^{-\frac{|x-a|}{b}} & \text{otherwise.} \end{cases} \quad (7)$$

The Laplace crossover, proposed in the paper [11], generates two offsprings $y^1 = (y_1^1, \dots, y_n^1)$ and $y^2 = (y_1^2, \dots, y_n^2)$ based on two parents $x^1 = (x_1^1, \dots, x_n^1)$ and $x^2 = (x_1^2, \dots, x_n^2)$ with the following procedure. First, a uniformly distributed random number $u \in [0, 1]$ is generated. Then, a number β is calculated:

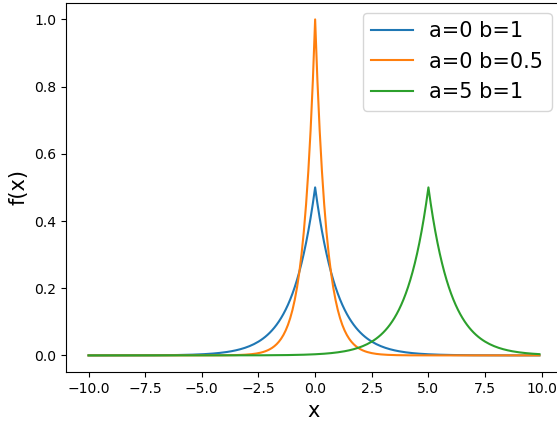
$$\beta = \begin{cases} a - b \cdot \log_e(u) & \text{if } u \leq \frac{1}{2} \\ a + b \cdot \log_e(u) & \text{otherwise.} \end{cases} \quad (8)$$

This formula for β is obtained by inverting the distribution function of the Laplace distribution. Finally, we compute the values of the offsprings like:

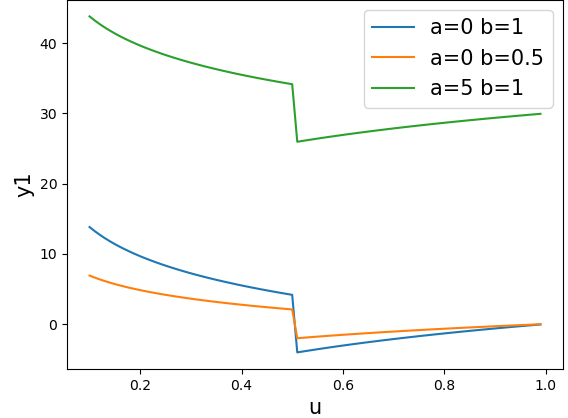
$$y_i^1 = x_i^1 + \beta |x_i^1 - x_i^2| \quad (9)$$

$$y_i^2 = x_i^2 + \beta |x_i^1 - x_i^2|. \quad (10)$$

The following graphs show on the left the Laplace distribution and on the right the possible values that can be reached for the offspring, if $x_1=0$ and $x_2=6$, according to equation 9



(a) Density function of Laplace distribution



(b) Values for the offspring y_1 for $x_1=0$ and $x_2=6$ depending on u

By looking at the Laplace distribution we see that the distribution is more concentrated when the value of b is smaller. The parameter a results in a shifting of the distribution. Because the way we construct the offsprings is based on this distribution, it is logical to see the same phenomenons on the right figure. For smaller values of b the offsprings are likely to be produced near the parents. For larger values of b the offsprings are expected to be produced far from the parents. Indeed, we see on the right figure that the range of reachable values is larger when b is larger. We also see that a induces a shift. Also, the values for the offsprings are proportional to the spread between the parents (see formula 9 and 10). Hence, if the parents are near to each other the offsprings are expected to be near to each other and if the parents are far from each other then the offsprings are likely to be far from each other. This way, the Laplace crossover operator show an adaptive behaviour. Indeed, in the beginning of search process the parents are very different (random initialization), while when the algorithm converges through a good quality region the parents are more and more similar. This characteristic is present in many operators as many use the difference between two solutions.

2.3.2 Heuristic crossover

The Heuristic crossover, proposed by Wright [20], generates one offspring $y = (y_1, \dots, y_n)$ based on two parents $x^1 = (x_1^1, \dots, x_n^1)$ and $x^2 = (x_1^2, \dots, x_n^2)$ with the following equation:

$$y_i = u(x_i^2 - x_i^1) + x_i^2, \quad (11)$$

where u is a uniformly distributed random number in the interval $[0, 1]$ and the parent x^2 is forced to have a fitness value not worse than the one of parent x^1 .

2.3.3 Non-uniform mutation

The non-uniform mutation is one of the most widely used mutation operators in real coded GAs. From a solution $x^t = (x_1^t, \dots, x_n^t)$, it creates a mutated solution $x^{t+1} = (x_1^{t+1}, \dots, x_n^{t+1})$ according to the following formula:

$$x_i^{t+1} = \begin{cases} x_i^t + (x_i^u - x_i^t)(1 - u^{(1-\frac{t}{T})^b}) & \text{if } r \leq 0.5 \\ x_i^t - (x_i^t - x_i^l)(1 - u^{(1-\frac{t}{T})^b}) & \text{otherwise,} \end{cases} \quad (12)$$

where t is the current generation number, T is the maximum number of generations, b determines the strength of the mutation operator, u and r are uniformly distributed random number between 0 and 1. In the initial generations, nonuniform mutation tends to search the space uniformly and in the later generations it tends to search the space locally, i.e. closer to the parents. We can see this phenomenon on the following plot (Figure 11) where all parameters and variable are constant. Indeed, starting from a point $x_i = 10$ we see that at time $t=0$ the mutation produce a point $x_i^{t+1} = 12.5$ and when t increase with the value of x_i^{t+1} stay closer to 10.

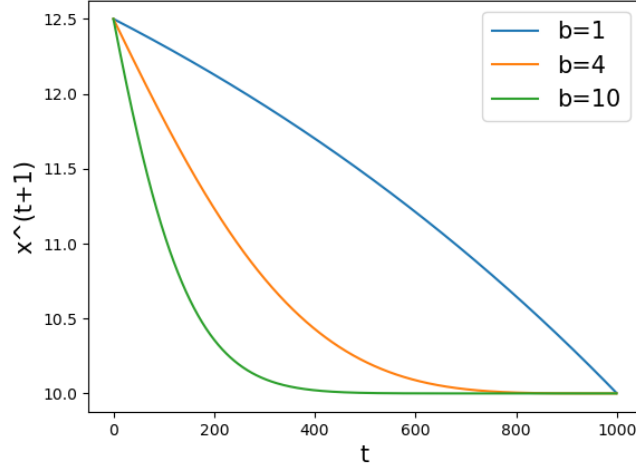


Figure 11: Evolution of the behavior of non-uniform mutation with time for $x^t=10$, $x_u=15$, $T=1000$, $u=0.5$, $r=0.6$

The idea behind this special behaviour is that in the beginning we are far from the optimum, thus we use mutation to explore the search space. While towards the end, the solutions in the population are close to the optimum and only small variation are required.

2.3.4 Power mutation

The Power mutation, proposed in [19], generates one offspring $z = (z_1, \dots, z_n)$ based on the vector $y = (y_1, \dots, y_n)$ with the following equation:

$$z_i = \begin{cases} y_i - s(y_i - y_i^l) & \text{if } t < r \\ y_i + s(y_i^u - y_i) & \text{if } t \geq r \end{cases}, \quad (13)$$

where y_i^l is the lower bound of variable i , y_i^u is the upper bound of variable i , $t = \frac{y_i - y_i^l}{y_i^u - y_i^l}$, r is a uniformly distributed random number between 0 and 1 and s follows a power distribution. The probability density function of the power distribution is expressed as follow:

$$f(x) = px^{p-1}, \quad 0 \leq x \leq 1. \quad (14)$$

The strength of mutation depends on the parameter p . Smaller values of p introduce less perturbation in the offspring and for large values of p more diversity is achieved. To illustrate the impact of p on the power distribution, here under is the the power distribution with with 3 values for p .

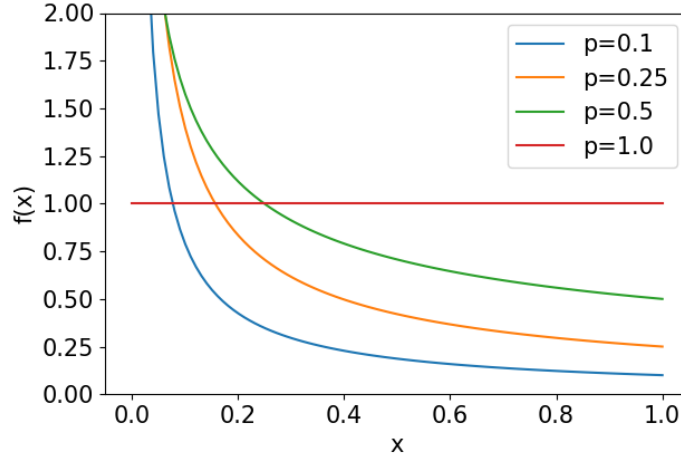


Figure 12: Power distribution for $p=0.1$, $p=0.25$, $p=0.5$ and $p=1$

We see that the smaller p is, the greater is the probability of choosing a small value for s . On the other hand, if p is larger it becomes more likely to reach value close to 1. Because s controls the perturbation introduced around y_i (see equation [13]), a small value of p introduce less perturbation. If we go to $p=1$ we see that it is similar to a uniform distribution.

Because those operators may produce solutions which are out of the variables domain an additional step is required. The strategy adopted is to attribute the upper bound value and the lower bound value respectively to the values greater than the upper bound and lower than the lower bound.

Selection technique for the Genetic Algorithm

In the paper [21], they found that ranking and tournament selection outperformed the other selection techniques for Genetic Algorithms when solving the TSP. In addition, the tournament selection is the chosen selection method in [11], [19] and [9], which have showed good results of Genetic Algorithms for a wide range of optimization problems including constraints and MILP optimization problems. The tournament selection is therefore chosen as the selection method for Genetic Algorithm in this work.

2.3.5 Summary GA

The GA algorithm implemented works with a tournament selection (see section 2.1.2), thus m tournaments are performed to select m solutions in the solution archive. The construction of new solutions:

Algorithm 1 construction m solutions with GA

Input:

m : number of selected solutions
 M : list of selected solutions
 cxp : crossover probability
 mp : mutation probability
 b : Laplace scale parameter
 p : strength of power mutation
 t : iteration number
 T : maximum number of iterations
 lb/ub : lower/upper bound
crossover/mutation : name of crossover/mutation

Output:

m new solutions

$i=0$

while $i < (m-1)$ **do**

```
    parent1, parent2 = M[i], M[i+1]
    rand1, rand2 are random numbers between 0 and 1
    if  $rand1 < cxp$  then
        if  $crossover == Laplace$  then
            (offspring1, offspring2) = Laplace(parent1, parent2, b, lb, ub)
        end
        if  $crossover == Heuristic$  then
            offspring1 = Heuristic(parent1, parent2, lb, ub)
            offspring2 = Heuristic(parent1, parent2, lb, ub)
        end
    end
    else
        (offspring1, offspring2) = (parent1, parent2)
    end
    if  $rand2 < mp$  then
        if  $mutation == Power$  then
            offspring1 = Power(offspring1, p, lb, ub)
            offspring2 = Power(offspring2, p, lb, ub)
        end
        if  $mutation == Non-Uniform$  then
            offspring1 = Non-Uniform(offspring1, t, T, lb, ub)
            offspring2 = Non-Uniform(offspring2, t, T, lb, ub)
        end
    end
    else
        pass
    end
    Store created offsprings,  $i += 2$ 
```

end

Once the solution are constructed we update the solution archive using the weak elitism technique.

2.4 Differential Evolution

Differential Evolution was proposed by Price and Storn in 1997 [22] and belongs, as GAs, to the family of evolutionary algorithms, but unlike GA, the original conception of DE is to handle continuous variables. The two operators used in DE are also crossover and mutation, but the mutation is performed before the crossover.

2.4.1 Differential mutation

The differential mutation creates a vector, called the mutant vector, denoted $m_j = (m_j^1, \dots, m_j^i)$. This mutant vector is obtained by operating on individuals of the population. Each individual of the population is denoted $x_j = (x_j^1, \dots, x_j^i)$. Where j is the index of the individual in the population $j \in [0, k]$ and i is the number of variables. This mutation operator will add, to a base vector, the difference between individuals. Here again different mutations are possible. To distinguish them, the following nomenclature is used: DE/a/b [7]. DE refers to differential evolution, a indicates how the base vector is chosen, and b is the number of differential vectors used in the mutation. The typical choice for a is "rand", which means that the base vector is chosen from the population using a uniform distribution; or "best", which means that the base vector is the best individual in the current population. For b, the two values are used, either 1 or 2 which means that we either do one or two differences. The three studied mutation in this thesis are DE/rand/1, DE/rand/2 and DE/best/1. The differential mutation will create a vector, called the mutant vector with the following equations.

DE/rand/1:

$$m = x_{r1} + F \cdot (x_{r2} - x_{r3}). \quad (15)$$

DE/rand/2:

$$m = x_{r1} + F \cdot (x_{r2} - x_{r3} + x_{r4} - x_{r5}). \quad (16)$$

DE/best/1:

$$m = x_1 + F \cdot (x_{r2} - x_{r3}). \quad (17)$$

Where, x_r is a vector chosen randomly from the population. All the vectors used to construct the mutant vector must be different. The vector x_1 is the best solution from the current population and $F \in [0, 2]$ is a scale factor which amplifies the differential variation among the vectors.

The mutation operator tries to combine good solutions to create even better solutions. Hence, it is mainly responsible for exploitation.

2.4.2 Differential crossover

The resultant vector of differential evolution is called the trial vector and it denoted $u^j = (u_j^1, \dots, u_j^n)$. The two most common differential crossovers are binary and exponential crossovers. The binary crossover is similar to the basic uniform crossover [9]. It performs an exchange between the variable of a mutant vector, v_j , and a selected vector, u_j . On the other hand, the exponential crossover is similar to the two points crossover [9]. The exponential crossover goes through the whole vector and for every variable it checks that a random number, r , is smaller than CR , the crossover rate [23].

Algorithm 2 Binary

```

 $v_j = x_j$ 
i=0
while  $i < n$  do
    r = random number between 0 and 1
    if  $r \leq CR$  then
         $v_j^i = m_j^i$ 
    end
    i+=1
end

```

Algorithm 3 Exponential

```

 $v^j = x^j$ 
i=rand(0, n-1)
r = random number between 0 and 1
while  $i < n$  and  $r \leq CR$  do
     $v_j^i = m_j^i$ 
    i+=1
    r = random number between 0 and 1
end

```

In differential evolution the crossover operator is used to promote the population diversity allowing more exploration [24].

Because those operators may produce solutions which are out of the variables domain a an additional step is required. The strategy adopted is to attribute the upper bound value and the lower bound value respectively to the values greater than the upper bound and lower than the lower bound.

2.4.3 Summary DE implementation

In the paper [7] which served as inspiration for the design of the differential algorithm, they apply the different operators to each of the solution from the solution archive. This is done by using the All population selection (see section 2.1.2). However, here an additional selection method is implemented to see the impact of updating the solution archive more frequently. The idea is that as we use solution vectors from the solution archive to construct a new solution, if the solution archive is updated more often with better solution vectors hence the construction of new solution might perform better and as a result the algorithm might converge faster.

To implement this more frequent update only a part of the population will be selected. To mimic the selection of the whole population here, at each iteration, we will select uniformly a sub-part of the population.

Hence if $m=k$ the all selection method is used and if $m < k$ the uniform selection is used.

The construction of the new solution follows the following algorithm (for DE/rand/1/Binary):

Algorithm 4 construct with DE

Input:

M : list of the selected solutions
m : number selected solutions
k : size of solution archive
SA : solution archive
DE_type : type of DE
cross_type : crossover type
F : scaling factor
CR : crossover rate

Output:

m new solution

```
for each solution X in M do
    r1 = random choice in [0, k-1]
    r2 = random choice in [0, k-1] \ {r1}
    r3 = random choice in [0, k-1] \ {r1, r2}
    r4 = random choice in [0, k-1] \ {r1, r2, r3}
    r5 = random choice in [0, k-1] \ {r1, r2, r4, r5}
    if DE_type == DE_rand_1 then
        | m = DE_rand_1(r1, r2, r3, SA, F)
    end
    if DE_type == DE_rand_2 then
        | m = DE_rand_2(r1, r2, r3, r4, r5, SA, F)
    end
    if DE_type == DE_best_1 then
        | m = DE_best_1(r2, r3, SA, F)
    end
    if cross_type == exponential then
        | new_sol = exp(m, X, CR)
    end
    if cross_type == binary then
        | new_sol = bin(m, X, CR)
    end
    Store created solution
end
```

Once a new solution is built based on a selected solution, this new solution is compared to the selected solution and, if there is an improvement in terms of fitness value, the new solution replaces the old one in the archive solution. The replacement method is performed for the m new solutions created.

2.5 Particle Swarm Optimization

The inspiration for Particle Swarm Optimization comes initially from the movement of organisms in a bird flock or fish school with particular characteristics. For example, bird flocks are able to, synchronously, change direction, scatter and regroup. E.O. Wilson made the following statement about fish schooling : “In theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches” [25]. This observation constitutes the inspiration for Particle Swarm Optimization (PSO).

The introduction of PSO algorithms dates back to 1995 by Kennedy and Eberhart. In PSO each solution is represented as the position of a particle and the particles evolve in a cooperative manner. A particle has a certain position and will evolve through the different iterations according to the its velocity vector. The position, $x_j = (x_j^1, \dots, x_j^i)$, and velocity vector, $v_j = (v_j^1, \dots, v_j^i)$, represent the particle’s evolutionary state in the search space. By iteratively changing those two vectors the particles evolves through the best position (the optimal solution). Unlike GA and DE, the PSO algorithms do not use evolution operators such as crossover and mutation.

In the classical formulation of PSO, the trajectory, of a population of particles, is determined by the following equations:

$$\begin{aligned} v_j(t+1) &= w \cdot v_j(t) + c_1 r_1 (\text{pbest}_j(t) - x_j(t)) + c_2 r_2 (\text{gbest}(t) - x_j(t)), \\ x_j(t+1) &= x_j(t) + v_j(t), \end{aligned} \tag{18}$$

where pbest_j is the personal best position searched by the j th particle and gbest is the best-so-far position searched by the whole swarm; c_1 and c_2 are two acceleration coefficients; r_1 and r_2 are two random numbers uniformly distributed between 0 and 1 and w is the inertia weight.

The first part is the momentum part, which forces the velocity to not change abruptly. The experimental analysis in [26] showed that, a high w is good for exploration while a small w favors exploitation. The second part is the “cognitive” part, which represents personal thinking by learning from its own experience (pbest). The third part is the social part, which represents the collaboration among particles by learning from group experience (gbest) [27]. The coefficients c_1 and c_2 maintain the trade-off between personal experience and social experience.

A common problem with this version of PSO is that, because all particles in the population learn from gbest , if the current gbest is far from the global optimum all particles are attracted to this sub-optimal region and might get trapped in a local optimum [28]. To improve PSO’s performance a possibility is to modify this gbest . A first way is to use different topologies to restrict gbest to a neighborhood. The following image illustrate some topologies used in the literature.

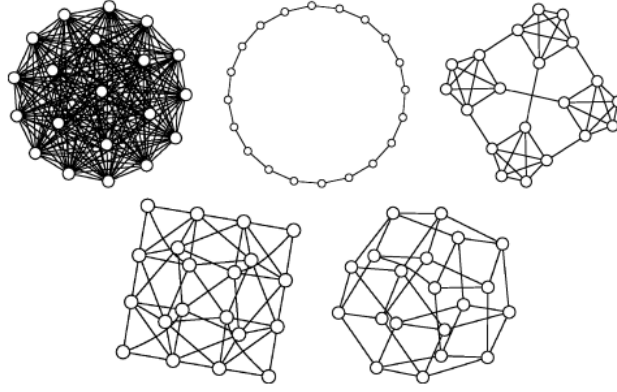


Figure 13: All, Ring, Four clusters, Pyramid and Square topology, From [29]

The particle chooses among its neighbours the one which has given the best solution vector. Thus, if we are in an All structure, the particle chooses among all the swarm. In classical particle swarm optimization a particle updates its velocity based on best position searched by the whole swarm. Hence, it corresponds to using an All structure. In the square square topology, each particle is connected to four other particles. Thus, gbest is the best solution among those four neighbours. The neighbourhood size is the number of particles to choose among and the higher it is, the better the one being picked is likely to be. Two other interesting statistics are the average distance, which measures the average number of edges between any two nodes and the diameter, which is the largest distance between two nodes in the graph. Those statistics, as well as the neighbourhood size, are used to describe the communication flow and the structure of the swarm. In the All topology, the average distance as well as the diameter is one, while in other topologies it is different of one and depends on the size of the swarm. Hence, in the All structure if a new gbest is found every other particle has access to it in the next iteration. It is not the case for the other topologies where there will be a delay depending on the communication flow. The four cluster represents four cliques which are connected to each other. We can make the comparison with four communities, where one communicates to another only via one person of the first community to one person of the second. The information will therefore take longer to spread within the communities. This allows to explore different search regions at the same time among the whole population. Many alternative structures are possible to mimic a desired behaviour. [29]

We can make a parallel between the structure and the concept of exploration versus exploitation. Indeed, the all structure favors exploitation because all the particles trajectory are influenced by the best solution, while the four cluster favors a bit more exploration because it allows to explore different local minimum at the same time.

It has been suggested that the classical version of PSO converges fast but has a tendency to converge to the local minimum while the versions with multiple "sub-swarm" might have more chances to find better solutions but more slowly [30].

The more "sub-swarm" there is, the more exploration we have. Hence, there is more chances to avoid getting stuck in a local minimum. However, increasing exploration can not guarantee good convergence. In order to promote exploration of swarm while maintaining good convergence efficiently, another possibility is to use the equation [19] for the velocity update (from [5]). This is the strategy used in this study as it is general and does not require any knowledge on the organisation of the swarm. Hence, the particle position (the solution) is updated as follow:

$$v_j(t+1) = w \cdot v_j(t) + c \cdot \text{rand}(\text{pbest}_r(t) - x_j(t)), \quad (19)$$

$$x_j(t+1) = x_j(t) + v_j(t), \quad (20)$$

where r is an index chosen randomly between 0 and $k-1$ (k is the population size) and pbest_r is the personal best position of the particle at index r . However, the randomly selected personal best, pbest_r , may be worse than the j^{th} particle. Thus the randomly selected particles may promote the diversity of swarm but deteriorate the convergence of swarm. To overcome this issue, we ensure that pbest_r must have a better fitness value than x_i .

In addition, the adaptive parameter tuning strategy from [5] is adopted to improve the overall efficiency of PSO. This adaptive tuning strategy allows to have different value of inertia weight, w , for different particles. These values evolve with iterations. The inertia weight will deviate from its initial value according to sometimes a Gaussian distribution and sometimes a Cauchy distribution (see algorithm below). Since the Gaussian distribution has thin tails on both sides the new computed value for w will be close its previous value, while the Cauchy distribution has larger tails and generate values with more exploration.

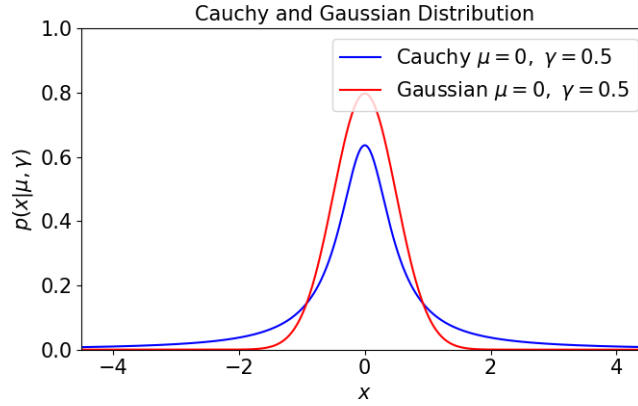


Figure 14: Cauchy vs Gaussian distribution

To balance exploration and exploitation, the probability of choosing the Gaussian or the Cauchy distribution is set to 0.5.

Algorithm 5 Tuning strategy of w

```

Generate a random number rand between
0 and 1
if rand < 0.5 then
    |  $w = \text{Cauchy}(\bar{w}, 0.1)$ 
end
else
    |  $w = \text{Gaussian}(\bar{w}, 0.1)$ 
end

```

The value of $\bar{w}$ and S is updated at each iteration only if there is an improvement in term of fitness of the new solution. Thus, if the new position vector $x^{\text{new},j}$ has a better fitness than previous position x^j then

$x^{new,j}$ replaces x^j in the solution archive and $S, \bar{w}$ are updated with the following equations:

$$S = S + 1, \quad (21)$$

$$\bar{w} = (\bar{w} * S + w) / (S + 1). \quad (22)$$

At the initialization step S is set to 0.

To help visualise the different variables, here is a representation of the population with the variables used in PSO.

Continuous Variables			Discrete Variables					
(Xr_1^1, Vr_1^1)	...	(Xr_1^r, Vr_1^r)	(Xo_1^1, Vo_1^1)	...	(Xo_1^o, Vo_1^o)	$f(X_1)$	$\bar{w}_1$	S_1
(Xr_2^1, Vr_2^1)	...	(Xr_2^r, Vr_2^r)	(Xo_2^1, Vo_2^1)	...	(Xo_2^o, Vo_2^o)	$f(X_2)$	$\bar{w}_2$	S_2
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
(Xr_j^1, Vr_j^1)	...	(Xr_j^r, Vr_j^r)	(Xo_j^1, Vo_j^1)	...	(Xo_j^o, Vo_j^o)	$f(X_j)$	$\bar{w}_j$	S_j
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
(Xr_k^1, Vr_k^1)	...	(Xr_k^r, Vr_k^r)	(Xo_k^1, Vo_k^1)	...	(Xo_k^o, Vo_k^o)	$f(X_k)$	$\bar{w}_k$	S_k

Figure 15: Representation of the population and the associated variables. Xr_j^i is a real decision variable, Xo_j^i it a discrete decision variable, Vr_j^i is the velocity variable associated to Xr_j^i , Vo_j^i is the velocity variable associated to Xo_j^i , $f(X_j)$ is the fitness of the particle at rank j based on its position vector, $\bar{w}_j$ and S_j are variables associated to the particle at rank j

Also, during the initialization step the velocity is chosen between the lower and upper bound of each real and discrete decision variable. This is to stay in an acceptable range of values. Indeed, if we don't impose a limit we risk to have velocities of too high order of magnitude.

Because those operators may produce solutions which are out of the variables domain a an additional step is required. The strategy adopted is to attribute the upper bound value and the lower bound value respectively to the values greater than the upper bound and lower than the lower bound.

2.5.1 Summary PSO implementation

In particle swarm optimization, the selection method is to select the all population. However, here an additional selection method is implemented to see the impact of updating the solution archive more frequently. The idea is that, if we update the solution archive more often a new *pbest* can be used more often too and therefore it might converges faster. This hypothesis is tested in the section [6](#).

To implement this more frequent update, only a part of the population will be selected. To mimic the selection of the whole population here, at each iteration, we will select uniformly a sub-part of the population.

Hence, if $m=k$ the all selection method is used and if $m < k$ the uniform selection is used.

The construction of new solutions respects the following algorithm:

Algorithm 6 construct with PSO

Input: M : list of size m of the selected solutions SA : solution archive c : acceleration coefficient $\bar{w}$: inertia weight**Output:** m new solutions

```
for each solution  $X$  in  $M$  do
   $V$  = associated velocity vector
   $w$  = tuning_algorithm( $\bar{w}$ )
  for each variable  $x_j^i$  of the particle do
     $r$  = random index,  $j \leq r \leq k$ 
     $pbest_r$  = particle's position in  $SA$  at index  $r$ 
    compute new velocity with equation 19 which depends on  $V$ ,  $c$ ,  $\bar{w}$ 
    compute new position with equation 20
  end
  Store new velocity and new position
end
```

Once a new solution is built based on a selected solution, this new solution is compared to the selected solution and, if there is an improvement in terms of fitness value, the new solution replaces the old one in the archive solution. The replacement method is performed for the m new solutions created. Additionally, if there is an improvement the value for $\bar{w}$ and the S are updated according to equations 22 and 21

2.6 Ant Colony Optimization

Ant colony optimization was first developed by Marco Dorigo in 1992 [31] and was the first swarm intelligence-based algorithm. It was inspired by the observation of the behavior of real ants. In many species, the ants walking to a food source deposit on the ground a substance called pheromone which influences other ants. The pheromones deposited on the ground form a pheromone trail, which allows the ants to find good sources of food that have been previously identified by other ants.

Initially, each ant randomly walks from the colony to the food source depositing pheromone. After a certain time, the shortest path will have more pheromones because in a determined amount of time a short path can be used more times than a long one. Secondly, ants tend in probability to follow a stronger pheromone trail which will accentuate the pheromone level on short paths.

In ACO, an artificial pheromone trail is used as a way of communication among the ants and as with real ants the pheromone trail is updated in the search process. These artificial pheromones are used to model a probability distribution of the different possible choices for a certain problem. For example, in TSP the different choices are the different routes between the cities. During the iteration, each ant chooses, according to a probability distribution, a route to follow. This choice will modify the pheromone level of the route and thus its probability to be chosen by another ant. The pheromone level acts here as an implicit memory. The central idea of ACO is thus a construction of solutions based on a probabilistic choice between the components of the solution.

This initial version is suited to solve combinatorial optimization problem as we can enumerate the different choices and assign a probability to each of them based on their pheromone level but when we are working with continuous variables this is not possible as each variable has an infinite number of possibility. The trick is to go from a discrete probability distribution, which is build based on implicit memory to a continuous probability density function, which is build based on an explicit memory. This explicit memory is the solution archive of all the solution vectors representing the solutions of all the ants in the population at each iteration.

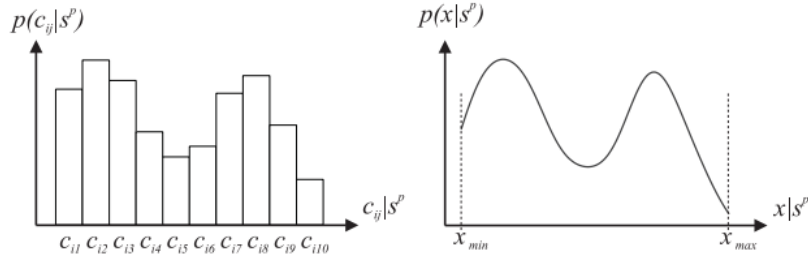


Figure 16: Distribution for discrete and continuous variable, from [32]

The proposed ACO in [6] to deal with real values uses a Gaussian distribution truncated between the lower and the upper bound. The Gaussian distribution is built for each decision variable of the selected solution and we sample it to construct the new solutions.

Gaussian distribution:

$$g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (23)$$

with mean μ and standard deviation σ :

$$\mu = R_j^i, \quad (24)$$

$$\sigma = \xi \sum_{l=1}^k \frac{|R_l^i - R_j^i|}{k-1}. \quad (25)$$

The σ value is thus computed as a parameter ξ multiplied by the average distance between the values of the i -th continuous variable of the solution j and the values of the i -th continuous variables of the other solutions in the archive [6].

2.6.1 Summary ACO

For this algorithm, the m solutions chosen among the solution archive are chosen using a rank selection.

The algorithm for the construction of a new solutions is as follow:

Algorithm 7 construct with ACO

Input:

M : list of size m of the selected solutions

SA : solution archive

k : population size

m : number of new solution to construct

Output:

m new solutions

for X *in* M **do**

 compute μ based on eq. [24] which take as input X

 compute σ based on eq. [25] which takes as input ξ , SA , k

 construct the new solution by sampling for each decision variable the Gaussian distribution [23]

 Store new solution

end

Once the m new solutions are constructed, we update the solution archive by keeping the best solutions among the k old solutions and the m new solutions (k -best update).

2.7 Adaptive Estimation of Distribution Algorithm

Estimation of Distribution Algorithm is another well-known metaheuristic. It extracts the global statistical information about the search space from the search so far. Then it builds a probabilistic model of promising solutions and a new solutions is created by sampling this [3]. It is thus very similar to the continuous version of ACO but the difference is that in EDA we construct one model from the population for each decision variable and then we sample it several times while in the continuous version of ACO we construct one model by decision variable and by ant that we sample only once.

The classical EDAs are partitioned in three classes [33]. The first category regroups the "Univariate EDAs" where it is assumed that all variables are independent. The most common one are PBIL, UMDA and cGA. The second category are the "Bivariate models" and they have the advantage to represent low order dependencies among the variables (MIMIC, BMMA and dependency tree-based EDAs). Finally, "Multivariate EDAs" allow to represent a higher level of dependencies than Bivariate models but are slower and require more memory (FDA, BOA).

There are many ways to estimate the distribution of a the decision variable. The chosen approach proposed in [8] and [34] and followed here is to use a certain part of the population, the NP_{best} solutions, to approximate the mean and the variance. Then a Gaussian or a Cauchy distribution is build based on the mean and variance of the NP_{best} solutions. The probability of choosing either the Gaussian or the Cauchy distribution is controlled by a parameter, g_p , which adapts itself during the iterations. This is why it is called an Adaptive Estimation of Distribution Algorithm. This strategy was proposed in [8] in order to avoid premature convergence. Indeed, the Cauchy distribution as larger tails (see figure [14]) and hence allow more exploration which as always goes with a slower rate of convergence. We also use here truncated distributions so that we stay in the bounds of the problem.

The parameter g_p changes from one iteration to another according to the following formula:

$$g_p = \left(\sum_{k=0}^1 \frac{top_d_k}{count_d_k} \right)^{-1} \frac{top_d_0}{count_d_0}, \quad (26)$$

where, top_d_0 and top_d_1 is the number of individuals sampled from Gaussian distribution and Cauchy distribution among the NP_{best} best individuals, respectively. $count_d_0$ and $count_d_1$ is the number of new solutions constructed by sampling from Gaussian and Cauchy distribution, respectively. To avoid the situation that the probability of choosing one distribution becomes too large or too small, the value of g_p is confined to the interval $[0.1, 0.9]$ [8].

The Cauchy and Gaussian distribution are built with a μ and σ computed as follow:

$$\mu_j^i = \frac{\sum_{j=0}^{NP_{best}-1} x_j^i}{NP_{best}}, \quad (27)$$

$$\sigma_j^i = \sqrt{\frac{\sum_{j=0}^{NP_{best}-1} (x_j^i - \mu_j^i)^2}{NP_{best}}}. \quad (28)$$

The main advantage of EDAs, over genetic algorithms for example, is the small number of hyperparameters that have to be tuned [33]. We only need to choose NP_{best} , the size of the population and the number of new solution created at each iteration.

2.7.1 Summary AEDA

In adaptive estimation of distribution algorithms, the solutions from the solution archive are the NP_best solutions.

Construction of m new solutions.

Algorithm 8 construct with AEDA

Input:

m : number of new solutions to construct
 NP_best
 SA : solution archive
list of used distribution for the solution in the solution archive

Output:

m new solutions
list of used distribution associated to the new solutions

```

for 1 to  $m$  do
  compute  $g_p$  based on the list of used distribution, computed
  by eq. (26)
  compute  $\mu$  based on  $SA$  and  $NP_{best}$ , computed by eq. (27)
  compute  $\sigma$  based on  $SA$  and  $NP_{best}$ , computed by eq. (28)
  if  $rand < g_p$  then
    | construct new solution by sampling Gaussian distribution
  end
  else
    | construct new solution by sampling Cauchy distribution
  end
  Store new solution with used distribution
end

```

Once the m new solutions are constructed, we update the solution archive by keeping the best solutions among the k old solutions and the m new solutions. If a new solution enters the solution archive at position j the value of the used distribution at position j must also be updated by the 0 or 1 depending on the distribution used to build this new solution.

3 Benchmark functions

This section introduces the role and the limitation of benchmarking as well as some rules for the design of a benchmark function. Additionally, five benchmark functions are detailed in terms of mathematical formulation and characteristics.

For every optimization algorithm we wish to understand their behavior, how quickly they converge as well as the accuracy of the solution found. However, as those metaheuristics do not rely on profound mathematical properties, it is impossible to answer these questions by only looking at the algorithm definition. This is the main reason why benchmark functions are used. Indeed, running algorithms on functions whose characteristics are known, such as the optimal solutions, allows us to better understand the functioning of these algorithms and to test how an algorithm performs in comparison with other algorithms. There are many existing benchmark functions that are proposed in the literature [35]. Here, to assess the performances of the algorithms, the chosen benchmark functions are : Ackley, Griewank, Rastrigin, Rosenbrock and Sphere. These five widely used benchmark functions respect the "rules for design" [36]:

- they are scalable with respect to the search space dimension,
- the design of problem instances should preferably exclude biases towards certain algorithm classes,
- the benchmarks essentially need to introduce measures of effectiveness, efficiency and variability
- they are reasonably quick to evaluate.

The five functions are indeed scalable and their mathematical definition is the following:

1. Ackley's Problem:

$$\min_x f(x) = -20 \exp \left(-0.02 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right) + 20 + e \quad -15 \leq x_i \leq 15,$$

$x^* = (0, 0, \dots, 0)$ and $f(x^*) = 0$.

2. Griewank Problem:

$$\min_x f(x) = 1 + \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos \left(\frac{x_i}{\sqrt{i}} \right) \quad -15 \leq x_i \leq 15, x^* = (0, 0, \dots, 0) \text{ and } f(x^*) = 0.$$

3. Rastrigin Problem:

$$\min_x f(x) = 10n + \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i)] \quad -15 \leq x_i \leq 15, x^* = (0, 0, \dots, 0) \text{ and } f(x^*) = 0.$$

4. Rosenbrock Problem:

$$\min_x f(x) = \sum_{i=1}^{n-1} \left[100 (x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right] \quad -15 \leq x_i \leq 15, x^* = (1, 1, 1, \dots, 1) \text{ and } f(x^*) = 0.$$

5. Sphere Function:

$$\min_x f(x) = \sum_{i=1}^n x_i^2 \quad -15 \leq x_i \leq 15, x^* = (0, 0, \dots, 0) \text{ and } f(x^*) = 0$$

All the five functions have the same value for global minimum which is zero, this facilitates the comparison of the performance on different benchmark functions. Moreover, the position of the global minimum is

different on the Rosenbrock function compared to the rest, to ensure that the algorithm is not "cheating" to find the optimal solution. Indeed, if all the global minimum were in position $x^* = (0, \dots, 0)$ then an algorithm which initialize all the variable to zero would always find the global minimum in 0 iteration. The chosen domain is $-15 \leq x_i \leq 15$ for each variable and a dimension of 9 is chosen. This is a good compromise between difficulty and time management. Indeed, for dimension 2 and 3 the problem are considered as easy to solve [37] while a dimension of 100 is very time consuming. Furthermore the real case has 10 variables.

The figure 17 represents the 5 benchmark functions in 3D (dimension = 2) to visualize the characteristics of each function.

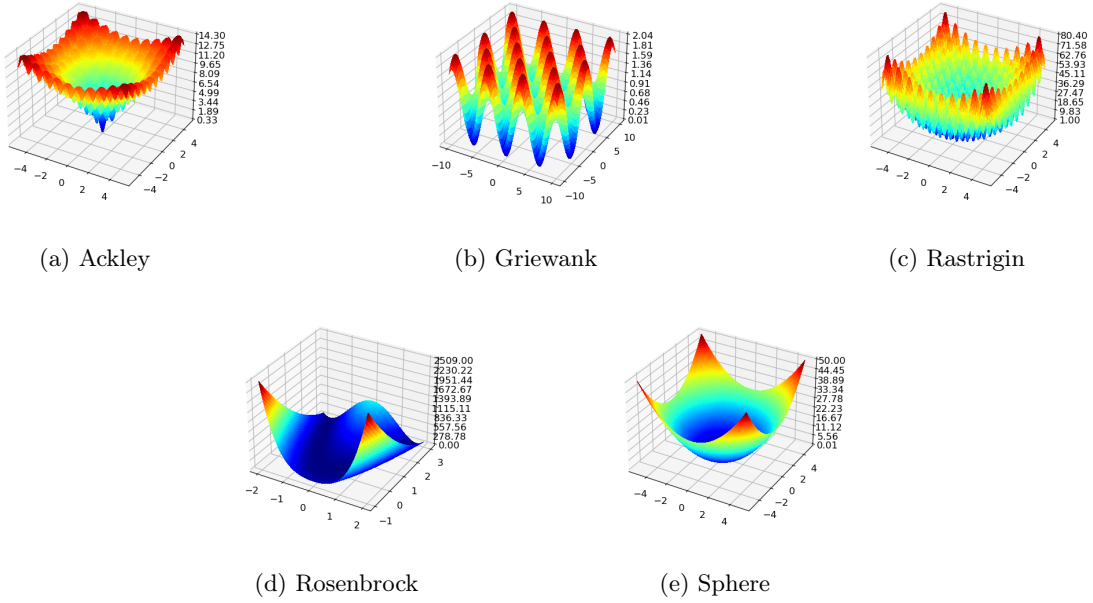
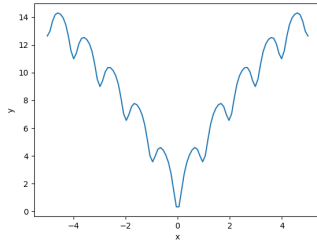


Figure 17: Benchmark functions in 3D

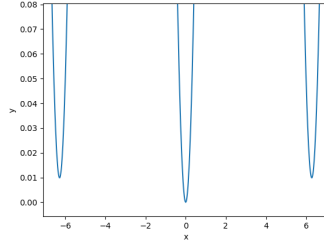
The idea behind working with different functions is that we don't always know the characteristics of the real case, thus these five benchmark functions provide information on the behaviour of algorithms for different characteristics. We can identify four possible characteristics: [37]

- Multi-modality (multim.) refers to the number of local optima of a problem
- Global to local optima contrast (gl.-loc) refers to the difference between global and local optima.
- Global structure (gl.-struc.) is the structure that remains after smoothing the local minima. For example, we see that the Rastrigin function (Figure 17c) has the same global structure as the Sphere function (Figure 17e).
- Sensitivity of a function, in this context, is the impact of a small change in the decision variable on the objective function.

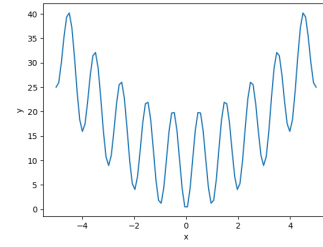
The higher the contrast between global and local minimum is, the easier it should be to solve the problem. Here is an illustration of the global to local contrast in the three multi-modal functions:



(a) Ackley



(b) Griewank



(c) Rastrigin

We see that the contrast in the Ackley function is much smaller than for the Griewank and Rastrigin function. It is thus expected to be easier to solve the Ackley function.

In terms of sensitivity the Rosenbrock function is very sensitive which means that small changes in the values of the decision variables are responsible in high changes in the function evaluation. We see on Figure 17d, that already with 2 decision variables on a small range $[-2, 2]$ and $[-1, 3]$ the function evaluation goes from 0 to 2509. On the other hand, the values for the Griewank function evaluation in Figure 17b goes from 0 to 2.04.

Function	multim.	gl.-loc	gl.-struc.	sensitivity
Ackley	high	high	Spherical	medium
Griewank	high	low	Spherical	very low
Rastrigin	high	low	Spherical	medium
Rosenbrock	none	none	non-Spherical	very high
Sphere	none	none	Spherical	medium

Limit of the benchmarking

Even though benchmarking is useful to perform a sensitivity analysis and analyze the behaviour of meta-heuristic algorithms, the over performance of one specific algorithm is not sufficient to encourage the use of this specific algorithm for all the optimization problems. Indeed, these benchmark functions are well designed and smooth, while real-world problems can be very diverse. In addition, these test functions are unconstrained while many real-world optimization problems have non-linear constraints. Finally, due to the NFL theorem, one algorithm which performed badly on the benchmark function may outperform the other algorithms for a specific application.

4 Real case problem

This section introduces the mathematical model of the hydrogen supply chain design, the situation in terms of demand and resources. The availability of green electricity is illustrated and the strategy used to solve this real case problem is described.

The case of interest, in this master thesis, is the design of the hydrogen supply chain design including production and transport technologies in Belgium. This is formulated as an MINLP optimization problem minimizing the total cost.

The case of interest considers here only the production of hydrogen thanks to water electrolysis. Water electrolysis is the process of water decomposition under the influence of a direct electric current which induces the redox reactions:



Hence, the carriers are hydrogen (H_2), oxygen (O_2), water (H_2O) and electricity (E). For what concerns the transport, only the use of hydrogen pipeline is considered. This choice of only considering pipelines is plausible as according to [38] it is the most cost-effective transport technology for distances below 1500km.

Here under are descriptions of the different sets, parameters and variables used in the model formulation.

Set	Description	Elements	
$\mathcal{H}$	Set of conversion technologies	Electrolysis	
$\mathcal{K}$	Set of transport technologies	Hydrogen pipeline	
$\mathcal{N}$	Set of nodes	BE21, BE32, BE33, BE34, BEp	
$\mathcal{C}$	Set of carriers	Electricity, Hydrogen, Water, Oxygen	
Parameter	Description	Domain	U.m.
$l_{k,i,i'}$	Distance between nodes i and i' for transport technology k	$\mathbb{R}_+$	km
$d_{j,i}$	demand of carrier j at node i	$\mathbb{R}_+$	MWh
p_h^{min}	minimum load of conversion technology h	$\mathbb{R}_+$	-
p_k^{min}	minimum load of transport technology k	$\mathbb{R}_+$	-
Variable	Description	Domain	U.m.
$S_{h,i}$	Capacity of conversion technology h at node i	$\mathbb{R}_+$	MW
$P_{j,h,i}$	Output flow of carrier j from conversion technology h at node i	$\mathbb{R}_+$	MWh/year
$G_{j,h,i}$	Input flow of carrier j into conversion technology h at node i	$\mathbb{R}_+$	MWh/year
$S_{k,i,i'}$	Capacity of transport technology k from node i to node i'	$\mathbb{R}_+$	MW
$F_{j,k,i,i'}$	Flow of carrier j through transport technology k from node i to node i'	$\mathbb{R}_+$	MWh/year
$U_{j,i}$	Import of carrier j into node i from the grid	$\mathbb{R}_+$	MW/year
$V_{j,i}$	Export of carrier j into node i from the grid	$\mathbb{R}_+$	MW/year
p_i	Pressure of hydrogen at node i	$\mathbb{R}_+$	bar

4.1 Objective function

$$J = \sum_{j \in \mathcal{C}} \sum_{h \in \mathcal{H}} \sum_{i \in \mathcal{N}} j_h(S_{h,i}, P_{j,h,i}) + \sum_{j \in \mathcal{C}} \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} j_k(l_{k,i,i'}, S_{k,i,i'}, F_{j,k,i,i'}) + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{C}} u(U_{i,j}) - \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{C}} v(V_{i,j}) + \text{penalty} \quad (29)$$

This objective function sums the production cost, computed by j_h , the transportation cost, computed by j_k , the importation cost of the carriers computed by u , the valorization of exportation computed by v and

a penalty is added when the constraint [37](#) is not satisfied. We can further subdivide j_h and j_k into their installation cost (capex), c_h and c_k , and operational cost (opex), o_h and o_k .

$$j_h(S_{h,i}, P_{h,i}) = c_h(S_{h,i}) + o_h(P_{h,i}) \quad (30)$$

$$j_k(l_{k,i,i'}, S_{k,i,i'}, F_{j,k,i,i'}) = c_k(l_{k,i,i'}, S_{k,i,i'}) + o_k(l_{k,i,i'}, F_{j,k,i,i'}) \quad (31)$$

As we only consider the production of hydrogen thanks to water electrolysis and the transport of hydrogen thanks to pipeline, the summation over j and h can disappear in the first part of the summation and the summation over j and k can disappear in the second term. Hence, we can write the objective function as follow:

$$J = \sum_{i \in \mathcal{N}} j_h(S_{h,i}, P_{j,h,i}) + \sum_{i \in \mathcal{N}} j_k(l_{k,i,i'}, S_{k,i,i'}, F_{j,k,i,i'}) + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{C}} u(U_{i,j}) - \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{C}} v(V_{i,j}) + \text{penalty} \quad (32)$$

4.2 Constraints

Mass balance

The design of a hydrogen supply chain must satisfy some constraints. The first one being the mass balance at each nodes i for each carrier j .

$$\sum_{h \in \mathcal{H}} (P_{j,h,i} - G_{j,h,i}) + \sum_{k \in \mathcal{K}} \sum_{i', i' \neq i} (F_{j,k,i',i} - f_k(F_{j,k,i',i}, l_{k,i,i'}) - F_{j,k,i,i'}) + U_{j,i} - V_{j,i} = d_{j,i}. \quad (33)$$

Which can be simplified as:

$$(P_{j,h,i} - G_{j,h,i}) + \sum_{i', i' \neq i} (F_{j,k,i',i} - f_k(F_{j,k,i',i}, l_{k,i,i'}) - F_{j,k,i,i'}) + (U_{j,i} - V_{j,i}) = d_{j,i}. \quad (34)$$

Where f_{pipe} computes the loss in the pipeline of the carrier j when going from i' to i . Indeed, as the size of hydrogen is nearly four times smaller than an iron atom there are some losses by diffusion. In the equation [34](#) the first term is the production of carrier j thanks to electrolysis. The second term considers the net import of carrier j from other nodes and as we only allow the transportation of hydrogen by pipeline pipeline between the nodes this term is equal to zero for the carrier electricity/water/oxygen. The third term is the net importation of carrier j from the grid. The whole summation on the left side is equal to the demand.

Weymouth equation

As we are considering a gas flowing through the pipeline it must respect the Weymouth equation which is stated as:

$$\dot{m}_{ii'}^2 = \frac{\pi^2 d_{ii'}^5}{16 f_{ii'} z_{ii'} R T_{ii'} l_{ii'}} (p_i^2 - p_{i'}^2) = \frac{\tilde{w}}{l_{ii'}} (p_i^2 - p_{i'}^2), \quad (35)$$

with, $f = 1.85 \cdot 10^{-3}$, $z = 1.03$, $T = 300 \text{ K}$, $R = 4124 \text{ [} \frac{\text{J}}{\text{kgK}} \text{]}$ and $LHV_{H_2} = 120 \text{ MJ/kg}$ thus:

$$\tilde{w} = \frac{\pi^2 d_{ii'}^5}{16 f_{ii'} z_{ii'} R T_{ii'}} = 4.715 \cdot 10^{-4} [\text{m}^3 \text{s}^2] = 4.715 \cdot 10^{-4} [\frac{\text{kg}^2}{\text{s}^2} \frac{\text{m}}{\text{bar}^2}]. \quad (36)$$

The pressure level is computed by the following non linear equation:

$$p_i^2 = \sqrt{p_{i'}^2 + \frac{l_{ii'} \dot{m}_{ii'}^2}{\tilde{w}}} \quad (37)$$

$l_{ii'}$ and $\tilde{w}$ are parameters of the model depending on the case considered and $\dot{m}_{ii'}$ is the mass flow rate and is computed based on the flow of hydrogen.

Conversion technology

The production of hydrogen via electrolysis must respect the performance of an electrolyzer which is non-linear:

$$P_{j',h,i} = \eta_{h,j,j'}(S_{h,i}, G_{j,h,i}, P_{j',h,i})G_{j,h,i}, \quad (38)$$

where $\eta_{h,j,j'}$ is the efficiency function of the electrolyzer to produce hydrogen (j') based on electricity (j).

If the electrolyzer is in operation ($y_{h,i} = 1$), the output flow is subject to a lower bound value different from zero and to an upper bound which is the installed capacity, otherwise ($y_{h,i} = 0$) it is constrained to zero.

$$p_h^{\min} y_{h,i} S_{h,i} \leq P_{j',h,i} \leq y_{h,i} S_{h,i} \quad (39)$$

Another constraint is the maximum potential installation of electrolyzer ($s_{h,i}^{\max}$):

$$S_{h,i} \leq s_{h,i}^{\max} \quad (40)$$

Transport technology

If the technology is in operation ($y_{k,i,i'} = 1$), the flow is subject to a lower bound value different from zero and to an upper bound which is the installed capacity, otherwise ($y_{k,i} = 0$) it is constrained to zero.

$$p_k^{\min} y_{k,i,i'} S_{k,i,i'} \leq F_{j,k,i,i'} \leq S_{k,i,i'} y_{k,i,i'} \quad (41)$$

Another constraint is the maximum potential installation of pipeline ($s_{k,i}^{\max}$):

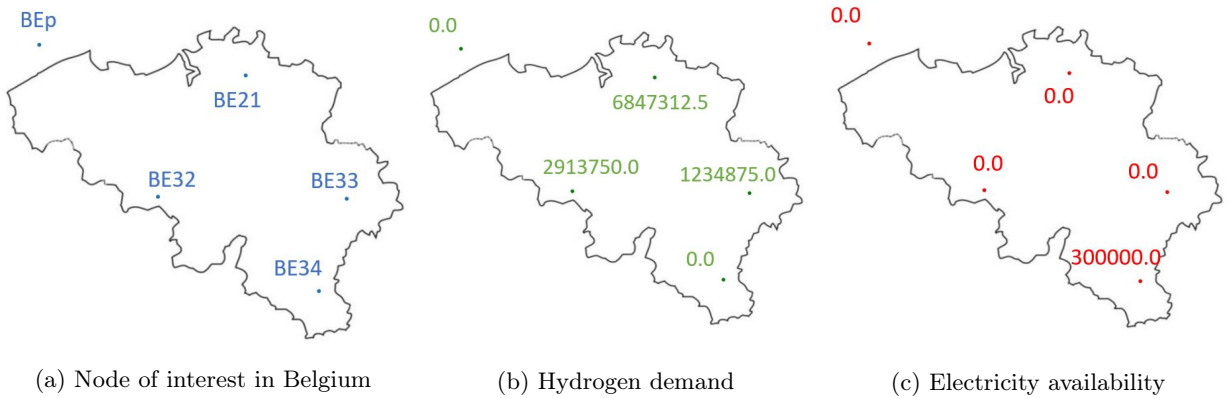
$$S_{k,i,i'} \leq s_{k,i}^{\max} \quad (42)$$

Availability

Finally, the importation of carrier from the grid must be smaller than the maximum availability:

$$U_{j,i} \leq a_{j,i} \quad (43)$$

The values of the demand and availability at each nodes for the hydrogen supply chain of Belgium are taken from [39].



4.3 Implementation

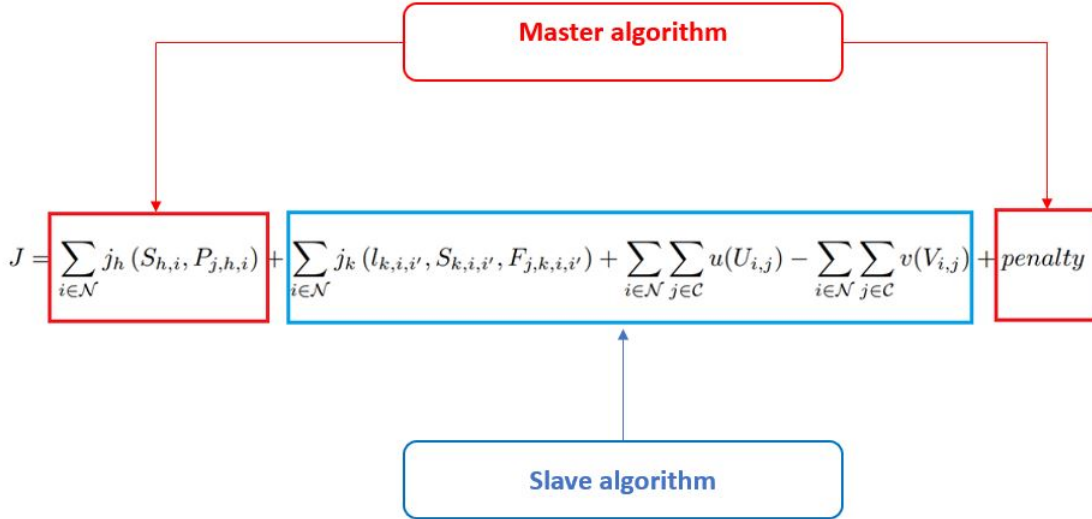
The strategy used to solve this optimization problem is to hybridize the metaheuristic algorithm by combining it with a linear solver. To do so, a master-slave architecture is used, where the master is the metaheuristic algorithm and the slave is the linear solver. The master algorithm will compute new solutions for the variables associated to the non-linear constraints and will also evaluate the respect of those non-linear constraints. The slave algorithm will then use the values of the variables computed by the master algorithm and consider them as fixed parameters to have a linear formulation.

At each iteration, the master algorithm computes new tentative solutions for the output flow of hydrogen, $P_{j,h,i}$ and the pressure at each nodes, p_i . Based on those values, some auxiliary variables can be computed. Indeed, knowing the output flow of hydrogen at each node allows to compute the required capacity installation of electrolyzer, $S_{h,i}$, as well as the required input flow of water and electricity $G_{j,h,i}$. The equation linking these variables [38](#) is non-linear. Hence, the need of computing it at the master level. We can also directly compute production cost $j_h(S_{h,i}, P_{j,h,i})$.

Then, the values of $P_{j,h,i}$, $S_{h,i}$, $G_{j,h,i}$ are passed as constants to the linear solver. This linear solver will then compute the importation and exportation of carrier, $v(V_{i,j})$ and $u(U_{i,j})$, and the flow of hydrogen in the pipelines $F_{j,k,i,i'}$ while respecting the above linear equations. This allows to compute transportation costs $j_k(l_{k,i,i'}, S_{k,i,i'}, F_{j,k,i,i'})$.

Based on the pressure at each node found by the master algorithm and the flow of hydrogen through the pipelines, we can evaluate degree of violation of the Weymouth equation [37](#). The higher the degree of violation, the higher the penalty added to the total cost. Finally, if the solution found does not respect a constraints (other than the Weymouth equation), the penalty is set to an very large value.

In summary, here is the repatriation of the optimization between the master and the slave algorithm.



5 Methodology

This section introduces the methodology used to make the choice among the possible GAs and DEs, among the hyperparameters and among the metaheuristics.

The strategy adopted in this study to orient our choices towards one specific instance of a metaheuristic algorithm is by analyzing their curve of convergence and the boxplot of the final solutions found by the metaheuristic. Due to the stochasticity, we cannot draw any interesting conclusion based on a single run. Hence, five runs are conducted when solving any optimization problem. The curve of convergence represents the mean among the five runs of the normalized best value found in the population for each function evaluation. The choice of the number of function evaluations for the x-axis is for a fair comparison of the metaheuristics and among the different instances of one particular metaheuristic. Indeed, if, for a fixed number of iterations, an algorithm updates the solution archive every five new solutions while another one updates it every 90 new solutions the number of function evaluations will be different. The relation between function evaluations and iterations is as follow:

$$\#iterations = (\#function_evaluations - k)/m, \quad (44)$$

where k is the size of the population and m is the number of new solutions constructed at each iteration. As the most time consuming part of metaheuristic is the function evaluation, using this criteria is provide a fair comparison.

For the choice of GA and DE, the sensitivity analysis and the comparison of tuned metaheuristic the metaheuristic algorithms are applied to the optimization of the benchmark functions, with 9 decision variables, presented in section 3 and the stopping criteria used here is the maximum number of function evaluations (FES_max) equal to $1e5$.

Methodology for the choice of the GA and DE

In order to choose the genetic algorithm that will be used in this study, the first step is to implement the crossover and mutation operators detailed in section 2.3. It is thus a question of implementing the Laplace and Heuristic crossover as well as the Power and Non-uniform mutation while respecting the structure described in section 2 and using a weak elitism update method. Once the implementation is done, the choice between the Laplace and the Heuristic crossover is done by comparing their results using a power mutation, as the power mutation had showed better results in the paper [19]. Then, the choice between the power and the non-uniform mutation is done using the previously chosen crossover and by comparing the results.

The values for the population size (k) is 10 times the number of decision variables and the number of the new solution created at each iteration (m) is equal to the the population size as recommended in the literature [21] [11]. The others hyperparameters are chosen based on their reference values.

The choice of the differential evaluation algorithm follows the same methodology in order to chose among DE_rand_1, DE_rand_2 and DE_best_1 as well as between exponential and binary crossover.

Methodology for sensitivity analysis

After implementing the five metaheuristics and choosing a specific GA and DE, the tuning of the hyperparameters is done to allow all the metaheuristics to perform at their best. As it would take too much time to evaluate each combination of hyperparameters to find the best one, the chosen approach is to chose the best parameter iteratively based on a sensitivity analysis. For example, if the goal is to find the best crossover rate and the best probability rate, first a sensitivity analysis is performed on the crossover rate which allows to understand the impact of this hyperparameter. Then, the crossover rate with the best performances is chosen when performing the sensitivity analysis on the mutation probability.

As a result of this strategy, some combination of hyperparameters are never tested. Furthermore, during the sensitivity analysis not every value is evaluated as the hyperparameters are real values and would require an infinite number of tests.

For every algorithm, some default hyperparameters are chosen based on the literature and these hyperparameters are updated one-by-one to find a combination of hyperparameters which gives good results.

As we iteratively choose the hyperparameters, here are the default ones necessary to start the tuning. Default hyperparameters for ACO [6]:

xi	q	m	k
0.6795	0.05099	5	90

Default hyperparameters for AEDA [8]:

NP_{best}	m	k
40(45% <i>ofk</i>)	90	90

Default hyperparameters for PSO taken from [5]:

c	m	k
2.0	90	90

Default hyperparameters for DE taken from [5] and [22]:

F	CR	m	k
0.2	0.2	90	90

Default hyperparameters for GA taken from [11], [19] for the hyperparameters for Heuristic crossover (HX), Laplace crossover (LX), Power mutation (PM), Non-uniform mutation (NUM) and for *cxp* and *mp* for the general observation over the literature that the *cxp* is often higher than *mp*.

HX-PM:

<i>cxp</i>	<i>mp</i>	<i>tr_size</i>	<i>elite_size</i>	PM strength	m	k
0.9	0.5	3	1	0.25	90	90

LX-PM:

<i>cxp</i>	<i>mp</i>	<i>tr_size</i>	<i>elite_size</i>	PM strength	LX strength	m	k
0.9	0.5	3	1	0.25	0.35	90	90

HX-NUM:

<i>cxp</i>	<i>mp</i>	<i>tr_size</i>	<i>elite_size</i>	NUM strength	m	k
0.9	0.5	3	1	4.0	90	90

LX-NUM:

<i>cxp</i>	<i>mp</i>	<i>tr_size</i>	<i>elite_size</i>	NUM strength	LX strength	m	k
0.9	0.5	3	1	4.0	0.15	90	90

Methodology for the choice of metaheuristic for the hydrogen supply chain design (HSCD)

For the selection of the most suited metaheuristic to solve the hydrogen supply chain optimization problem, we work with 10 decision variables. Indeed, the metaheuristic algorithm is responsible of finding solution for the output flow and the pressure at each node. Hence, as we are working on the case of 5 nodes in Belgium, 10 decision variables are necessary. The implementation of the algorithm is based on the code

developed by the Risk and Reliability Engineering Laboratory at ETH Zurich, which solves the HSCD as describe in section 4.

To determine the best metaheuristic algorithm, we start by performing an analysis of the behaviour of the five metaheuristics by plotting their convergence curve. For this plot, we use a maximum number of function evaluations of $3e4$ as termination criterion. Based on the behaviour, we eliminated the algorithms which don't have a good convergence curve. Then, we run again the promising algorithms five times with this time a stopping criterion based on the maximum number of stagnation iterations, and with the utilisation of restart strategy described in section 2. Here too, for fairness reasons, the maximum number of stagnation iterations is adapted to each algorithm by the same reasoning as the one described in the beginning of this section. The best final solution as well as the number of function evaluations before convergence are showed on boxplots. Based on the convergence curve and the boxplots, the best candidates are kept into consideration and the impact of the discretization is evaluated on all of them. Indeed, the decision variables are real variable but one could be interested in considering them as discrete variables. The discretization of the real variables could improve the speed of resolution as less possible values are considered. However, the discretization also leads to approximations. Hence, studying the impact on discretization is of a certain interest. Moreover, it serves as a confirmation that the metaheuristic implemented are indeed able to deal with discrete variables. To evaluate the impact of discretization we consider different levels of discretization and we run the algorithms with the maximum number of iterations criterion and the restart strategy. The level of discretization is the number of possible values for the discrete variables different of the minimum value. Hence, with a discretization level of 2 the possible values are : min, mean, max. The discretization level considered are: 2, 10, 100 and 100 000.

6 Results

This section presents the results of the comparison of possible GAs and DEs, then it presents the tuning done with the sensitivity of the metaheuristics to their hyperparameters. Then, it shows the comparison of the tuned algorithms on benchmark functions as well as on the design of hydrogen supply chain design. Finally, the impact of discretization is evaluated.

6.1 Choice of algorithm for GA

In order to design a Genetic Algorithm, one must choose the crossover and the mutation operators. Two of each were presented in the section 2.3. The analysis of their respective performances is presented here.

Choice of Crossover

The two types of crossover are the Heuristic and the Laplace crossover. The performances in terms of convergence as well as final result are presented here under.

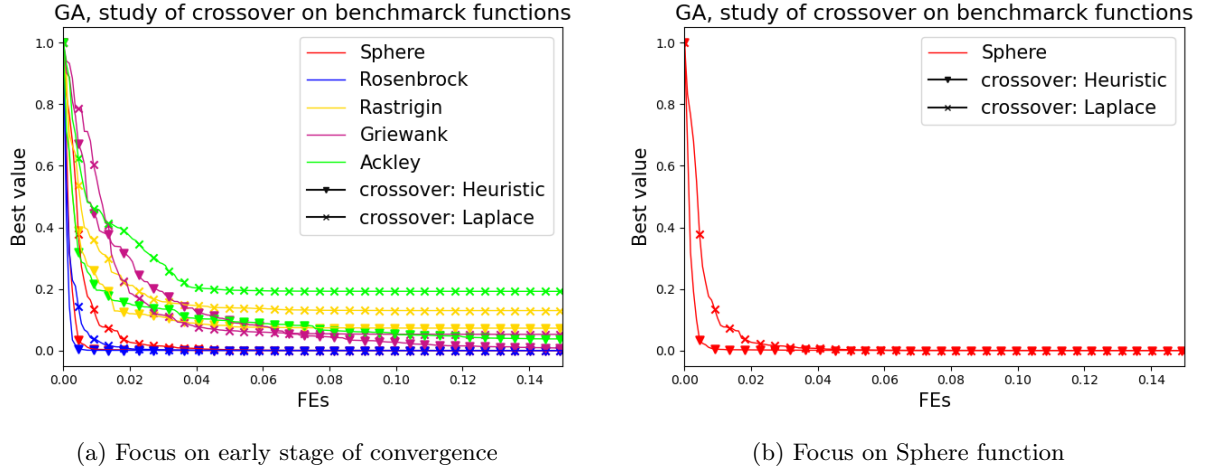


Figure 21: Curve of convergence of GA depending on crossover type on benchmark functions

We notice that the Laplace crossover converges slowly with respect to the Heuristic crossover, except on the Griewank function where the convergence behaviour is similar. If we look specifically on the sphere function, we realise that the Laplace crossover struggles more than the Heuristic one.

In the boxplot of the final results (Figure 22) we see that the Heuristic crossover performs better, or similarly, than the Laplace crossover.

As a result of a better convergence curve and better final result the Heuristic crossover is chosen.

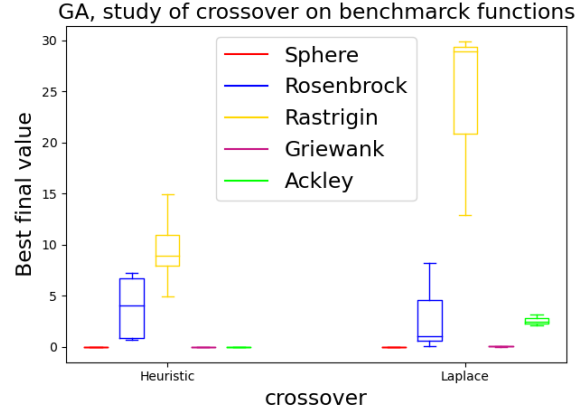


Figure 22: Boxplot for different crossover type on benchmark functions over 5 runs

Choice of Mutation

The two types of mutation are the Power and the Non-uniform mutations for which the performances in terms of curve of convergence as well as final result are presented here under.

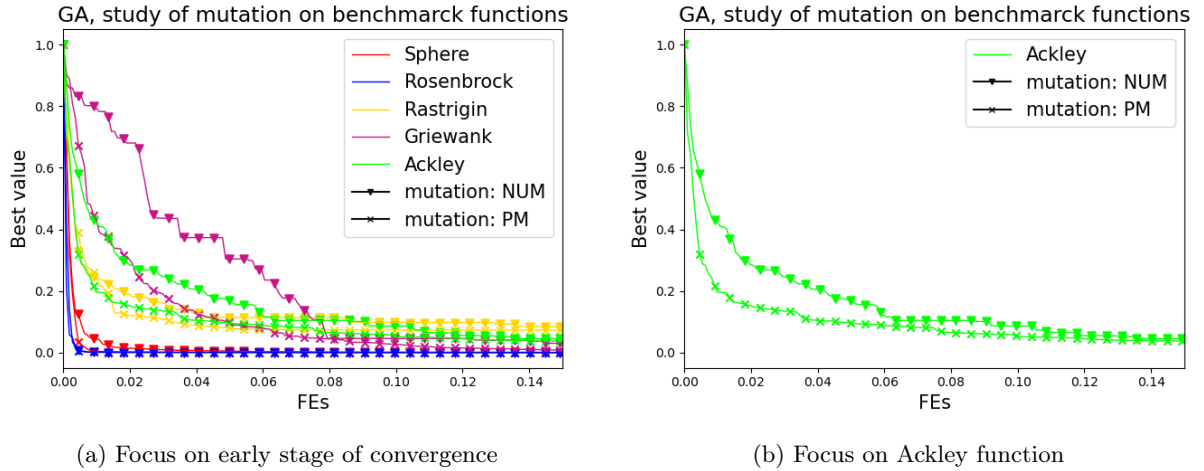


Figure 23: Curve of convergence of GA depending on mutation type on benchmark functions

On Figure ??, we clearly see that the Power mutation presents a better curve of convergence on all the benchmark functions.

In terms of final results, the power mutation outperforms the non-uniform mutation on the Rosenbrock function, while it is the opposite on the Rastrigin function. However, the power mutation manages to find the same value for the Rastrigin function one time among the five runs. The function evaluation of the Rosenbrock function using a non-uniform mutation never reach the minimum found by the power mutation. Hence, the power mutation is chosen as the mutation operator for GA.

An additional argument in favor of the power mutation is based on the way both operators work (see section 2.3). Indeed, the non-uniform mutation requires the maximum number of function evaluations (FEs_max), while when we use the maximum number of stagnation iterations as the stopping criterion, FEs_max might not be specified.

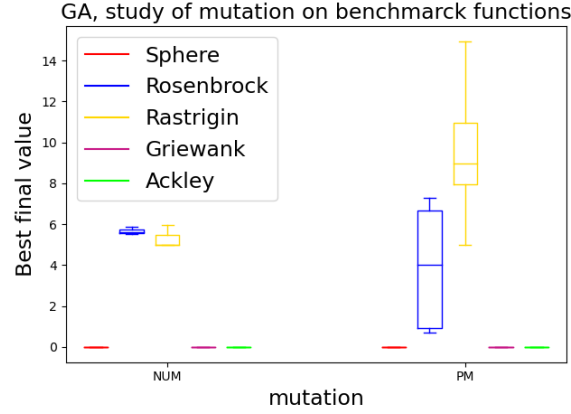


Figure 24: Boxplot for different mutation type on benchmark functions over 5 runs; NUM= non-uniform mutation, PM = power mutation

6.2 Choice of Algorithm for DE

Similarly to GA, some variations of Differential Evolution exist. It is therefore necessary to make the choice of the type of DE as well as the type of crossover.

Choice of the differential mutation <DE_type>

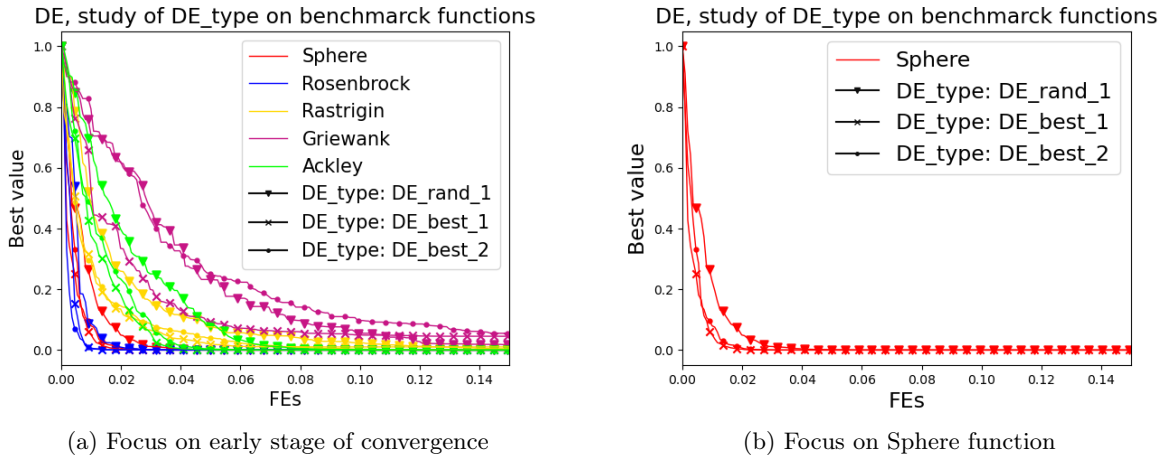


Figure 25: Curve of convergence of DE depending on DE_type on benchmark functions

The behaviour of the three types of DE depends on the function to minimize. However, we can see on the above Figure that DE/best/1 and DE/best/2 tend to converge faster than DE/rand/1. On the Griewank function we observe that the DE/best/2 doesn't show an appealing convergence curve.

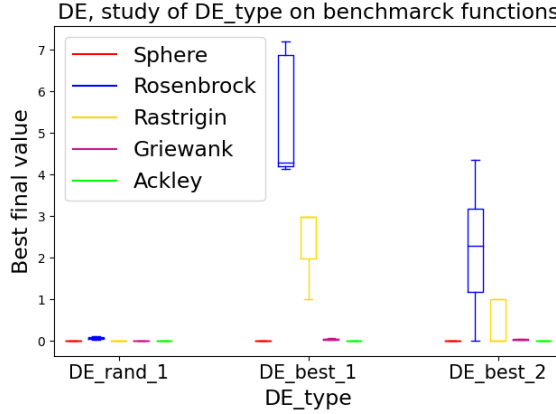


Figure 26: Boxplot for different DE type on benchmark functions over 5 runs

In terms of final result, there is no doubt that the DE/rand/1 outperforms the others. Hence, even though using the best solution to create new solutions allows to converge a bit faster, the DE/rand/1 is chosen for final results.

Choice of the crossover <cross_type>

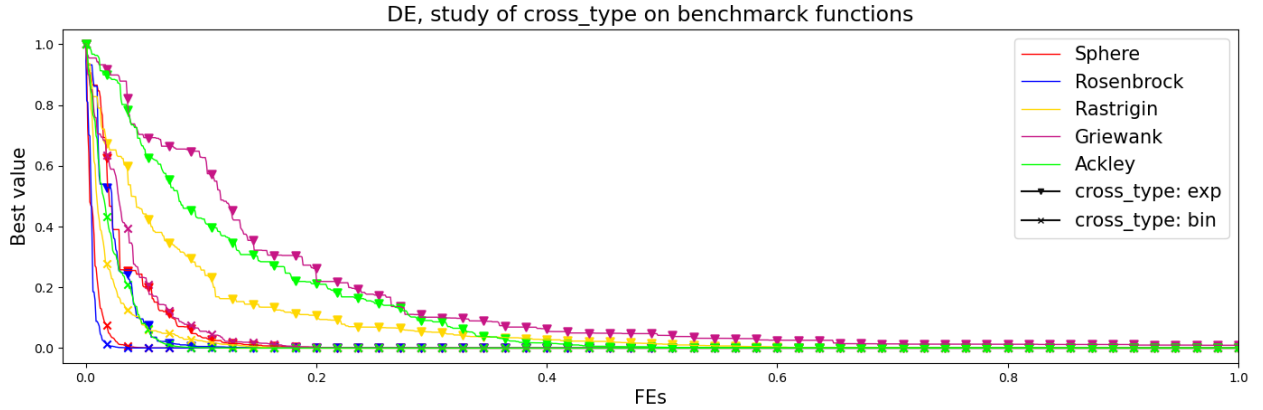


Figure 27: Curve of convergence of DE depending on cross_type on benchmark functions

We comparing the curve of convergence of the exponential crossover to the one of binary crossover, we see a much faster convergence for the binary crossover. Furthermore we see on Figure 28 that the binary crossover finds better or equal results on all the benchmark functions. Thus, the binary crossover is chosen as the crossover operation for the DE algorithm.

Better results on the Rosenbrock function can be explained by the fact that, both with exponential and binary crossover the algorithm still has not converged after $1e5$ function evaluations. Hence, because the binary converges faster it shows better results.

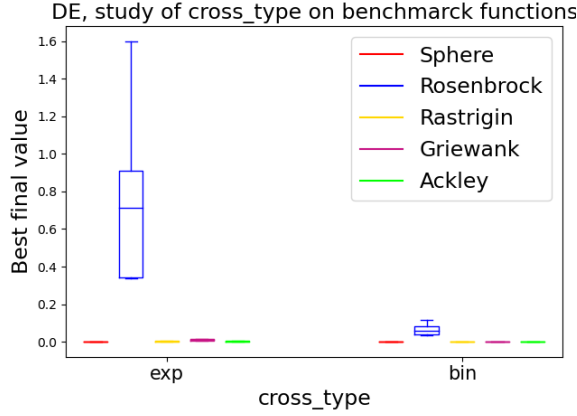


Figure 28: Boxplot for different DE crossover type on benchmark functions over 5 runs

6.3 Sensitivity Analysis of hyperparameters specific to a metaheuristic

Each metaheuristic algorithm has different hyperparameters which need to be tuned. The tuning of these hyperparameters is based on the sensitivity analysis performed in this section.

6.3.1 ACO

Choice of the parameter influencing the computation of the variance $\langle xi \rangle$

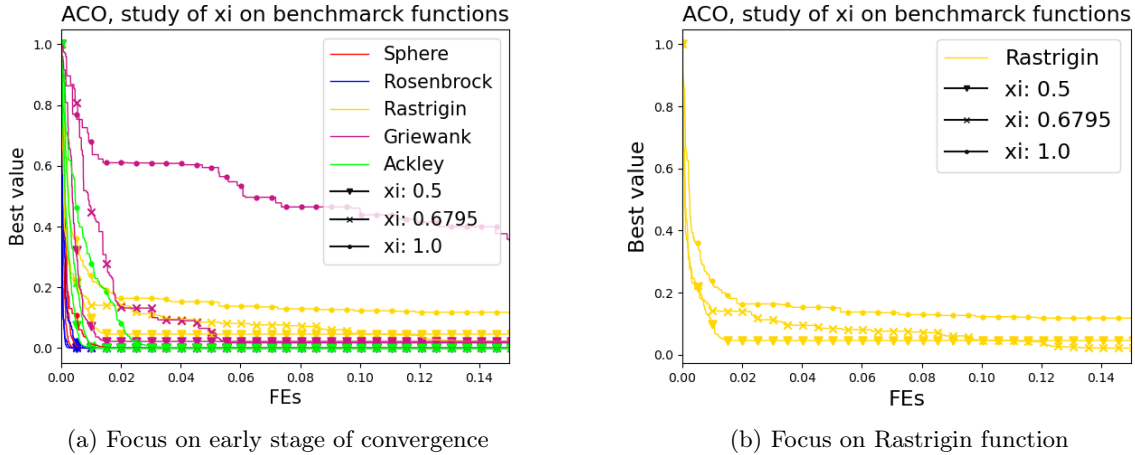


Figure 29: Curve of convergence of ACO depending on ξ on benchmark functions

First we see that the general tendency is for higher ξ to converge slowly. We especially see that, the convergence is slower for the value $\xi=1.0$ to solve the Griewank function. Furthermore, if we closely look at the behaviour on the Rastrigin function we see that it converges faster with $\xi=0.5$. But around FEs=10% of the maximum number of function evaluations the curve with $\xi=0.6795$ crosses the curve of $\xi=0.5$ resulting in a better final result as showed in the boxplot on Figure 30. We can understand this behaviour by remembering that ξ influence the variance. If the variance of the Gaussian distribution is small the algorithm is going to search closely around the selected solution while if the variance is large it will introduce more exploration of the search space. It is thus a compromise between exploitation and exploration. Small ξ prioritize exploitation while large ξ prioritizes exploration. For easy problem as

the Sphere function a small value is better as it will converge faster, while finding the optimal solution. However, when looking for the global minimum in a function with many local minima and a high contrast between local and global minimum, like the Rastrigin function, a higher ξ is to prefer. Another point of interest is that if ξ is too large the algorithm will not have the time to converge.

On Figure 30 shows that the best values are obtained with $\xi=0.6795$ on all the benchmark functions.

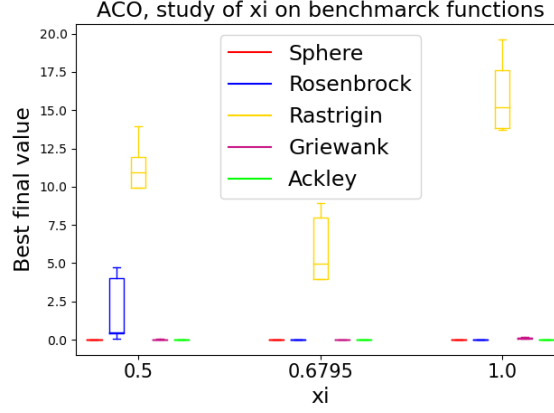


Figure 30: Boxplot for different ξ type on benchmark functions over 5 runs

Given that the advantage in terms of convergence is not very pronounced when comparing $\xi=0.5$ with $\xi=0.6795$, and because the best results are with 0.6795, this value is chosen for the hyperparameter ξ .

Choice of the parameter influencing the rank selection $\langle q \rangle$

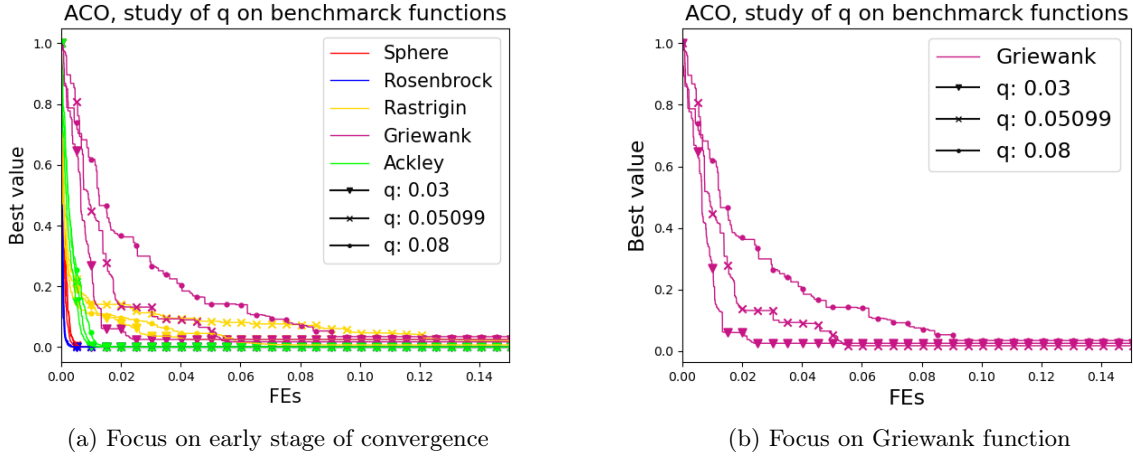


Figure 31: Curve of convergence of ACO depending on ξ on benchmark functions

A first observation is that the rate of convergence doesn't change much when varying q from 0.03 to 0.08 on four of the benchmark functions. However, we see that on the Griewank function, it converges faster with $q=0.03$ and $q=0.05099$ than with $q=0.08$. To explain this phenomenon we recall that q influence the weight (thus the probability to be selected) given for each solution in the solution archive (Figure 5). Indeed, the

smaller q is the more importance is given to the best solutions which is similar to favoring exploitation and resulting in a higher convergence rate. And, favoring exploitation often leads to a faster convergence rate.

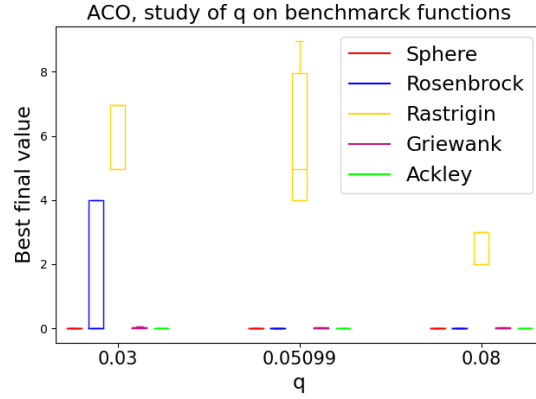


Figure 32: Boxplot for different q on benchmark functions over 5 runs

Nevertheless, as always there is a compromise to make. We see that for the Rastrigin function, a value of 0.03 for q is not enough as it doesn't allow enough exploration and get stuck in a local minimum. To back up this statement, the solution found for one of the five runs with $q=0.03$ is $(0, -0.99, 0, -0.99, 0.99, 0.99, 0, 0, 0.99)$, which correspond to a local minimum. Indeed, if we remember the figure 18c of the Rastrigin function with one variable we see that there is a local minimum in $x=1$. If we look at the solution found for one of the five runs with $q=0.08$ we get $(1, 0, 0, 0, 0, 0, 0, 0, 0)$. Hence, with a higher q we are able to escape more local minima and find a better solution. Hence, the value $q=0.08$ is chosen.

6.3.2 AEDA

Choice of the number of best solution used when creating new solutions $\langle NP_{best} \rangle$

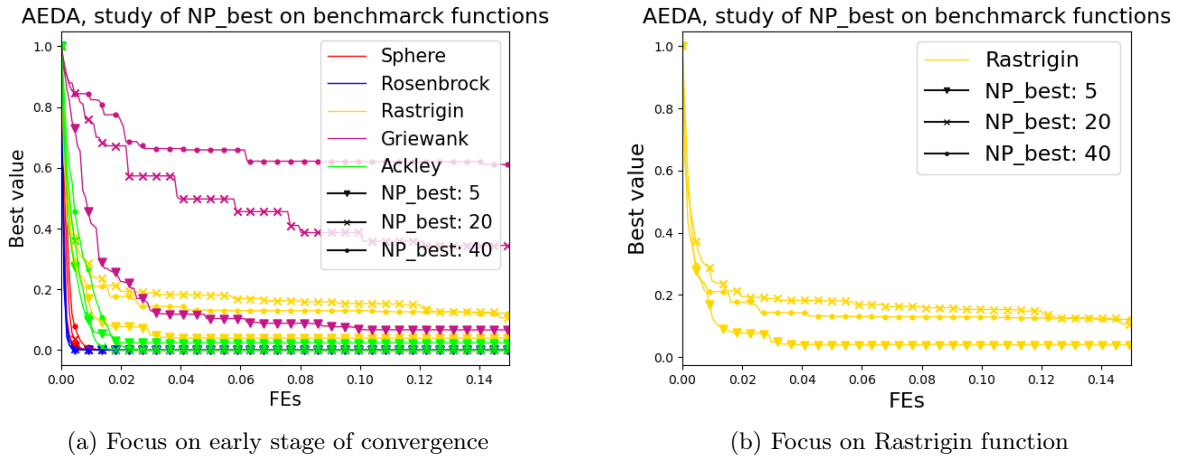


Figure 33: Curve of convergence of AEDA depending on NP_{best} on benchmark functions

The general behaviour is a faster convergence rate for small value of NP_{best} . This again reflect the compromise between exploration and exploitation. In AEDA, a small (resp. large) value of NP_{best} is to choose

if we mainly want exploitation (resp. exploration). As a result of that we see that on figure 35b at 14% of the maximum number of iteration the best result are achieved by $NP_{best} = 5$ then 20 and then 40.

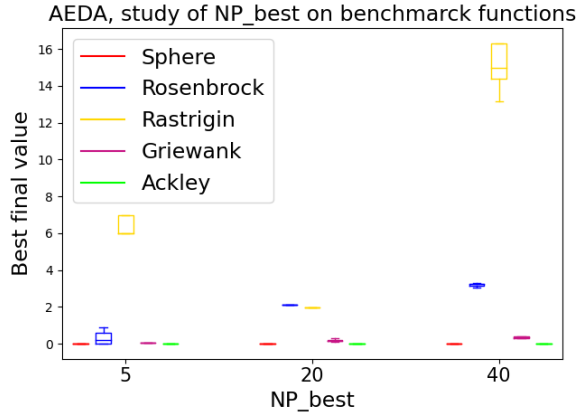


Figure 34: Boxplot for different NP_{best} on benchmark functions over 5 runs

First, we see that for the Sphere and Ackley function, all three values of NP_{best} find the global minimum. Then, on the Rosenbrock function we see that as NP_{best} increase the precision decrease. This is because the Rosenbrock function does not have local minimum and thus require more exploitation than exploration. The quality of the results on the Griewank function also decrease when NP_{best} increases, even though the Griewank function has many local minima and thus require exploration. To explain it, first we recall that the Griewank function is not very sensitive in the sense that, even for a solution vector that is not optimal the function evaluation will be close to the optimum. It explains why we could think that with $NP_{best} = 5$ we find the minimum while actually the best found solution vector is $(-3.14, 0, 0, -6.27, 0, 7.6, -8, 0, 0)$. Thus, it doesn't find the global minimum with $NP_{best} = 5$. As for why it is worse when allowing more exploration, we can understand it by looking at the curve of convergence. Indeed, the convergence rate is much too slow. Hence, it doesn't converge after $1e5$ FEs. Finally, the Rastrigin function first improve then get worse which again is the trade-off between exploration and exploitation. First it improves because the Rastrigin function need exploration (to avoid local minima) then it get worse because it has not enough time to converge. The chosen value is $NP_{best} = 5$.

6.3.3 PSO

Choice of the acceleration coefficient $<c>$

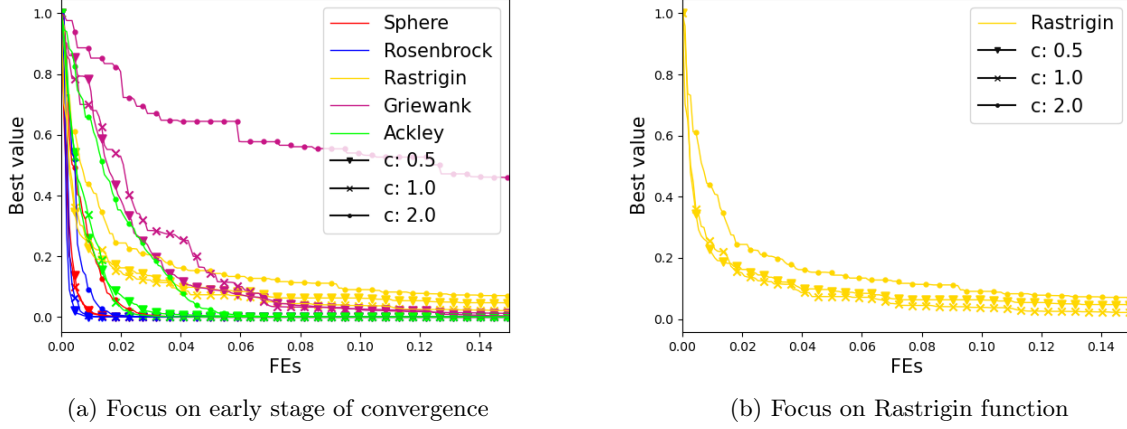


Figure 35: Curve of convergence of PSO depending on c on benchmark functions

We observe on Figure 35 that, for a large c the PSO algorithm converges more slowly. We also see that, the rate of convergence of $c=0.5$ and $c=1.0$ are very similar.

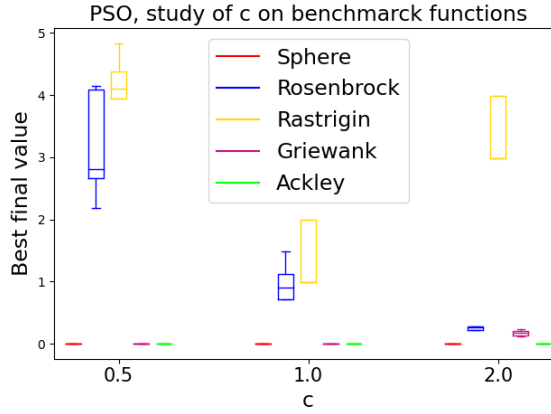


Figure 36: Boxplot for different c on benchmark functions over 5 runs

In terms of final results, the best performances are achieved by $c=1.0$ on four of the five functions. We only get a slightly better results on the Rosenbrock function for $c=2.0$. Hence, as we have a nice convergence curve and the best final results for $c=1.0$ this value is chosen in this study.

6.3.4 GA

Choice of the crossover probability $\langle \text{cpx} \rangle$

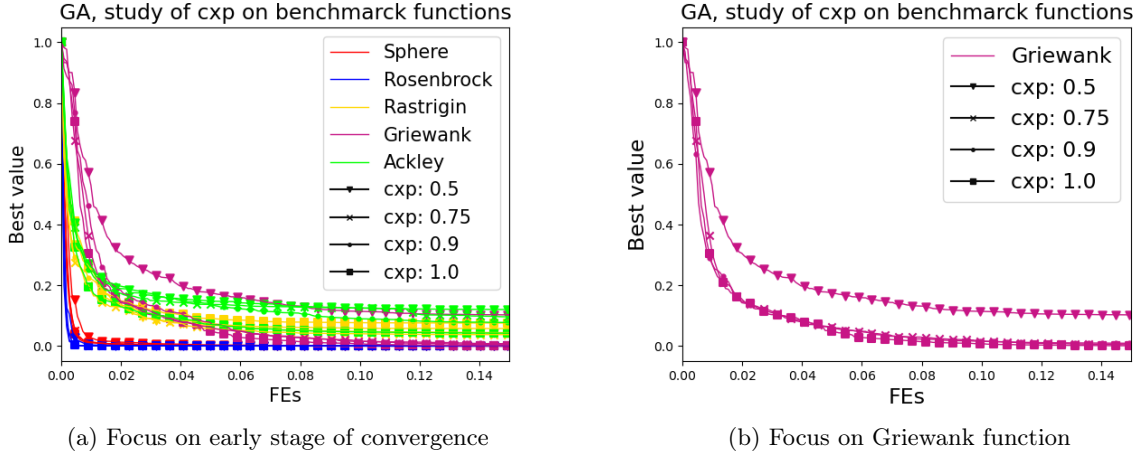


Figure 37: Curve of convergence of GA depending on cpx type on benchmark functions

In genetic algorithm, the crossover probability influences fraction of parents that will undergo the crossover operator. A crossover rate of 1 (resp. 0.5) means that the crossover is applied to 100% (resp. 50%) of the parents. By looking at the convergence curves, we see that, with a high crossover probability (0.75, 0.9 and 1) the Genetic Algorithm converges faster than for $\text{cpx}=0.5$. This can be attributed to the fact that the crossover operator is responsible for exploitation of the search space (see section 2.3). And, more exploitation lead to a higher convergence rate.

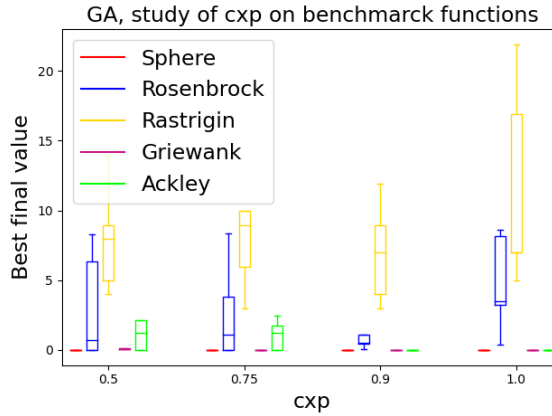


Figure 38: Boxplot for different cpx on benchmark functions over 5 runs

In terms of final results, we get similar boxplots for the four considered crossover probability. Nevertheless, the genetic algorithm has more trouble finding the global minimum for $\text{cpx}=0.5$ and $\text{cpx}=0.75$ than the $\text{cpx}=0.9$ or $\text{cpx}=1.0$. Moreover, $\text{cpx}=0.9$ performs better than $\text{cpx}=1.0$ on Rosenbrock and Rastrigin. Here, the choice of cpx is not obvious. The value $\text{cpx}=0.5$ is not chosen due to its slow speed of convergence and the value 0.9 is chosen as a compromise between 0.75 and 1.0.

Choice of the mutation probability $\langle mp \rangle$

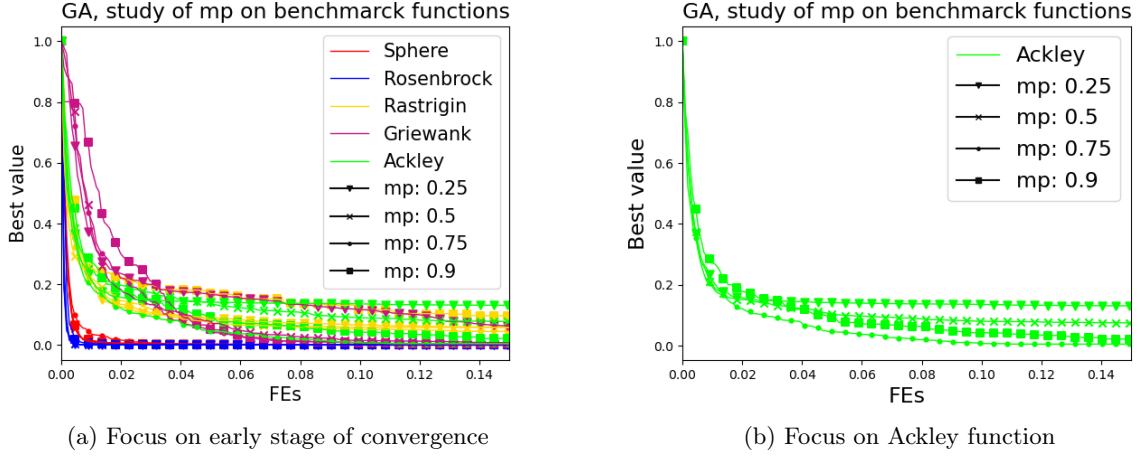


Figure 39: Curve of convergence of GA depending on mp type on benchmark functions

On the very early stage of convergence (up to 0.02), for the Ackley function, all the curves of convergence behave in a similar fashion. Then, for small value of mutation probability the algorithm tends to stagnate. If we recall that the mutation operator in GAs influences the exploration of the search space, this observation make sense as it allows to escape some local minima. We see that, a mutation probability of 0.25 is not enough, on the Ackley function, to escape the local minima. The low impact on the very early of the convergence rate can be explained by the fact that we have an initial population of size equal to ten times the dimension. Thus, we have a lot of possible good value to exploit with the crossover operator. It is suggested in [40] that, for small initial population size the mutation operator has more influence as it promote new solutions.

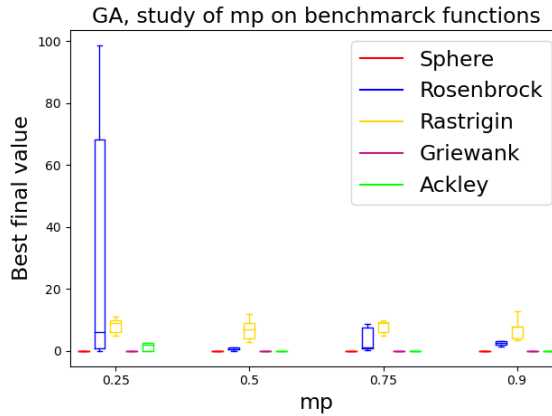


Figure 40: Boxplot for different mp on benchmark functions over 5 runs

On Figure 40 we see quite bad results for $mp=0.25$ compared to the other values. The final result with a mutation probability of 0.5 are slightly better and this value will be chosen for this algorithm.

Choice of strength of the power mutation $\langle \text{StrengthPM} \rangle$

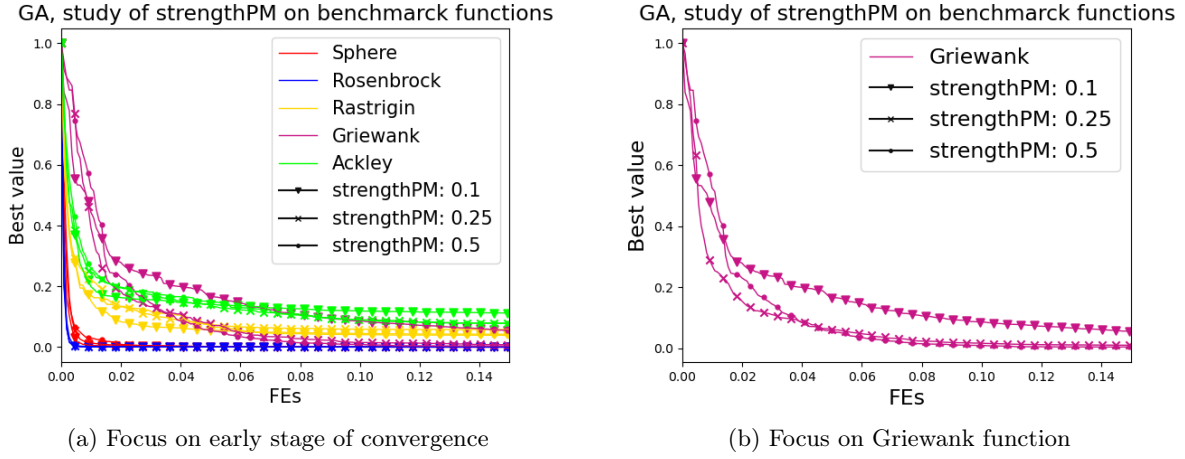


Figure 41: Curve of convergence of GA depending on StrengthPM type on benchmark functions

The strength of the power mutation influence the perturbation introduced around a solution vector. The higher the strength is the more likely it is to introduce an important perturbation (see section 2.3). Here three different values are considered: 0.1, 0.25 and 0.5. In terms of convergence, the difference is not very marked between those three values.

On the boxplot Figure 42 we see that, the algorithm doesn't always reach the global minimum when minimizing the Ackley function with a strength of 0.1.

The chosen value for the Power mutation strength is 0.25 as it gives the best result on Rosenbrock function and similar ones on the other function as well as a similar convergence curve.

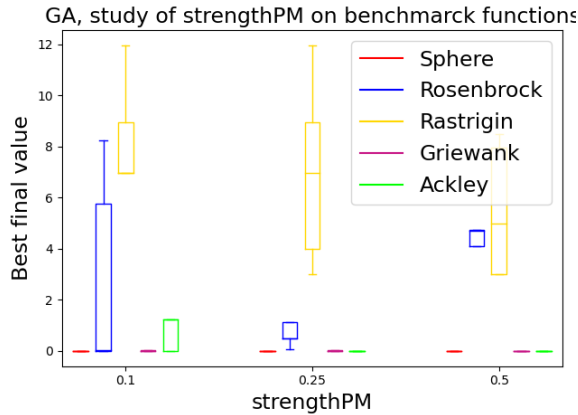


Figure 42: Boxplot for different strength for power mutation on benchmark functions over 5 runs

Choice of the tournament size $\langle tr_size \rangle$

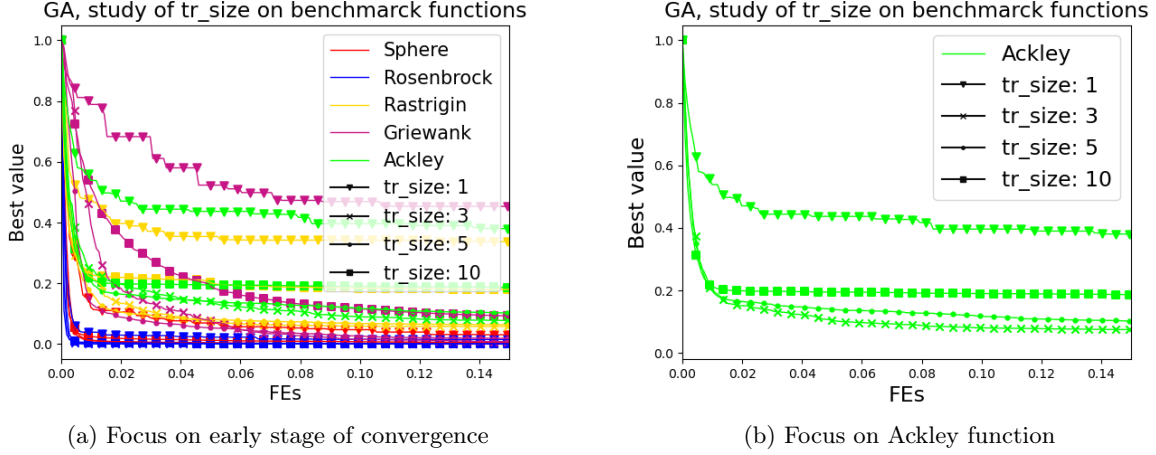


Figure 43: Curve of convergence of GA depending on tr_size type on benchmark functions

As showed in the figure 6, the size of the tournament selection influence the selection pressure of the algorithm. A high tournament size results on a high selection pressure towards the fitter individuals. A global observation is that a tournament size of one, which is equivalent to a random selection, doesn't exercise enough selection pressure. Therefore, it takes a long time to converge. When the genetic algorithm is used with a tournament size of 3, 5 or 10 the convergence rate is highly improved. Nonetheless, when focusing on the Ackley function we realise that, the algorithm converges too fast to a non-optimum value for $tr_size = 10$. A tournament size of 1 favors to much the exploration by choosing randomly among the population and result in a slow convergence speed, while a tournament size of 10 doesn't explore enough and get stuck in a non-optimum solution.

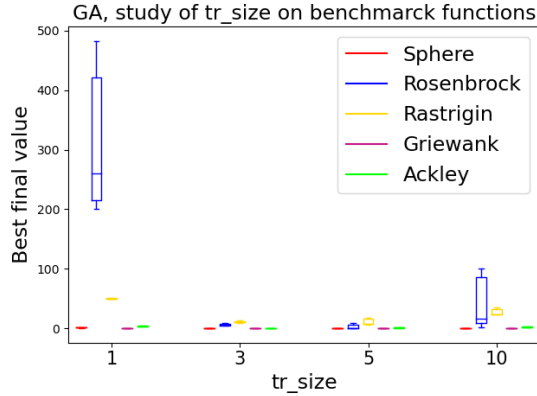


Figure 44: Boxplot for different tournament size on benchmark functions over 5 runs

A tournament size of 3 and 5 look like a good trade-off. Hence, the value of 3 is chosen as it is also the one recommended in the paper 9.

Choice of the elite size <elite_size>

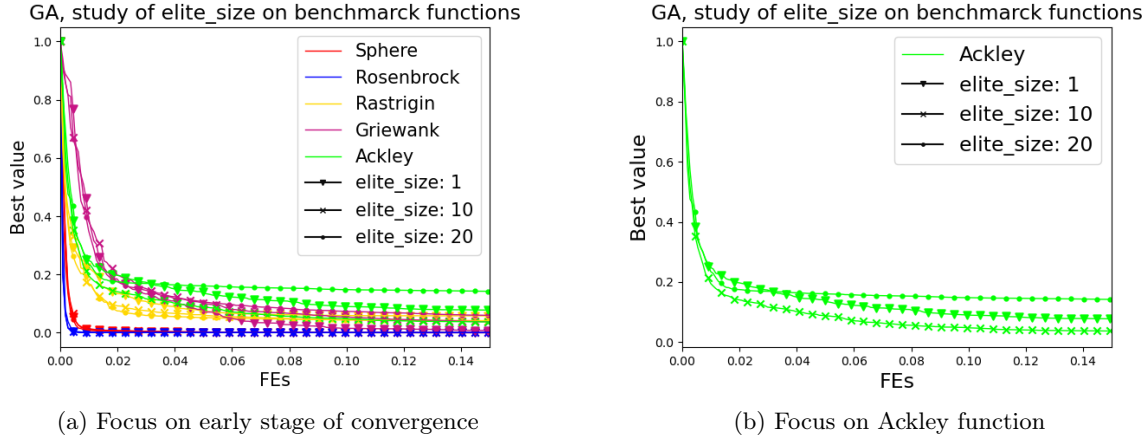


Figure 45: Curve of convergence of GA depending on elite_size type on benchmark functions

The elite size represent the number of best solution kept from one iteration to another. In the very early stage of convergence, the impact on the elite size is negligible. This phenomenon can be explain by the fact that in the very early step of genetic algorithm all the new created solutions, thanks to crossover and mutation, presents a better fitness than the old ones. Hence, the old solution are replace due to the weak elitism update strategy. After the very early stage of convergence, we see on the Ackley function that having an elite size of 20 result in a premature convergence.

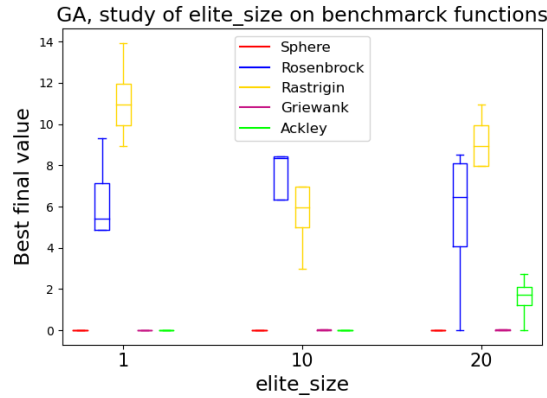


Figure 46: Boxplot for different elite size on benchmark functions over 5 runs

Based on the boxplot of the final result an elite size of 10 is chosen in this study as it performs well on the Sphere/Rastrigin/Griewank/Ackley function.

6.3.5 DE

Choice of the scaling factor $\langle F \rangle$

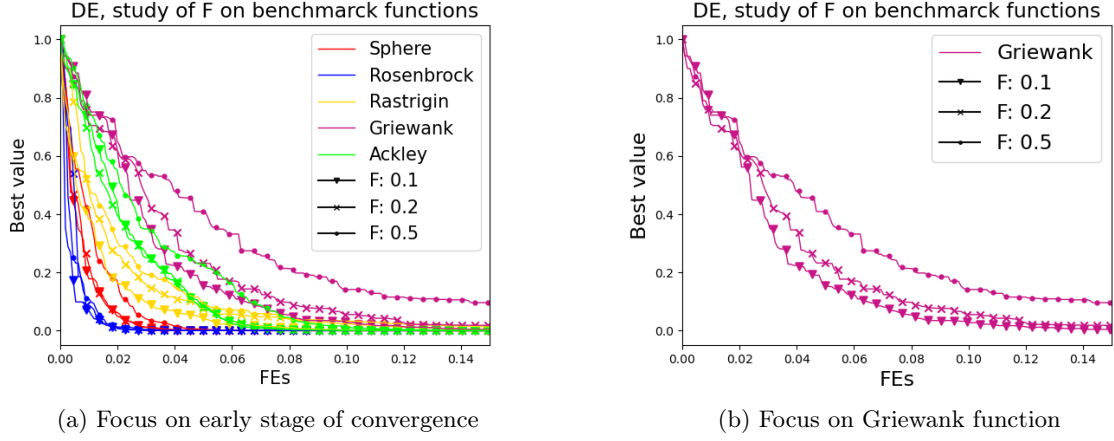


Figure 47: Curve of convergence of DE depending on F type on benchmark functions

F is the scaling factor in differential evolution which multiplies the difference between two solutions vector. We see on figure 47a and 47b that the higher F is, the slowly the algorithm converges. 47b where the algorithm has not converged at 15% of the maximum number of function evaluation.

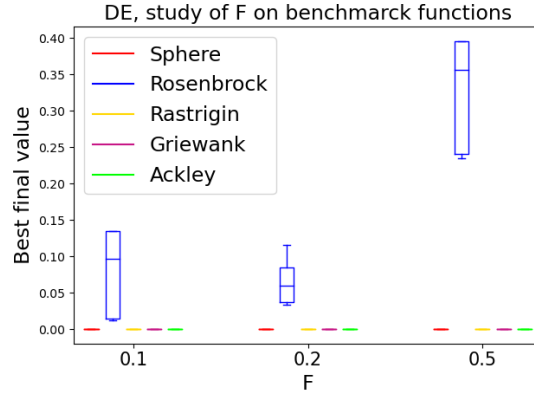


Figure 48: Boxplot for different F on benchmark functions over 5 runs

In terms of final result, the differential evaluation algorithm reaches the optimum on four of the five functions for each tested value of F . Because of the better convergence rate with $F=0.1$ we get a better final result with this value compared to the rest. Indeed, after $1e5$ iterations, the algorithm has not converged with all the three values of F . The value $F = 0.1$ is chosen for its faster convergence speed.

Choice of the crossover rate $\langle CR \rangle$

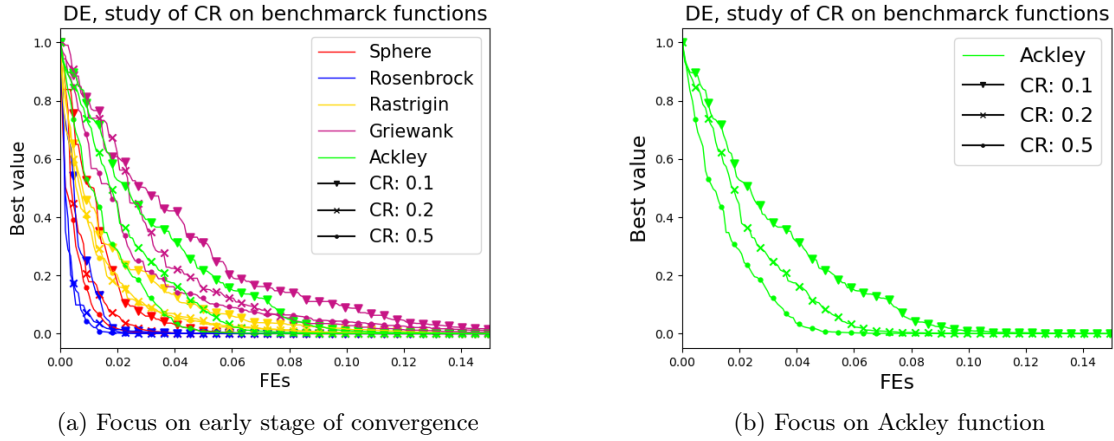


Figure 49: Curve of convergence of DE depending on CR type on benchmark functions

The crossover rate, CR, influences the degree at which we learn from the mutant vector. A crossover rate of 0.1 means that 10% of the variables in the new constructed solution comes from the mutant vector and 90% comes from the selected solution from the solution archive. We see the above figures that the higher the crossover rate is responsible for faster converge speed.

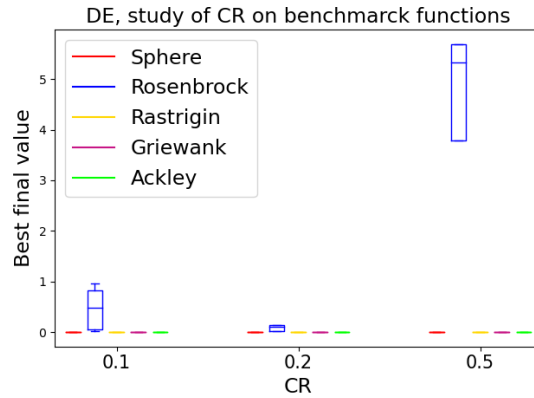


Figure 50: Boxplot, for different CR on benchmark functions over 5 runs

When looking at the final results, the differential evolution finds the optimum in four of the five benchmark functions. On the Rosenbrock function the best values are with a crossover rate of 0.2 which is therefore the chosen value.

6.3.6 Sensitivity Analysis of common hyperparameters

All the metaheuristics are population-based and iterative. Hence, for all of them the population size (k) as well as the number of new solutions created per iteration (m) need to be chosen.

Choice of the number of new solutions per iteration $\langle m \rangle$

The hyperparameter m influence the update rate of the population. The ACO, AEDA, PSO and DE algorithm have been adapted in order to allow working with different update rate while the genetic algorithm, due to its update strategy, only work with m equal to the population size.

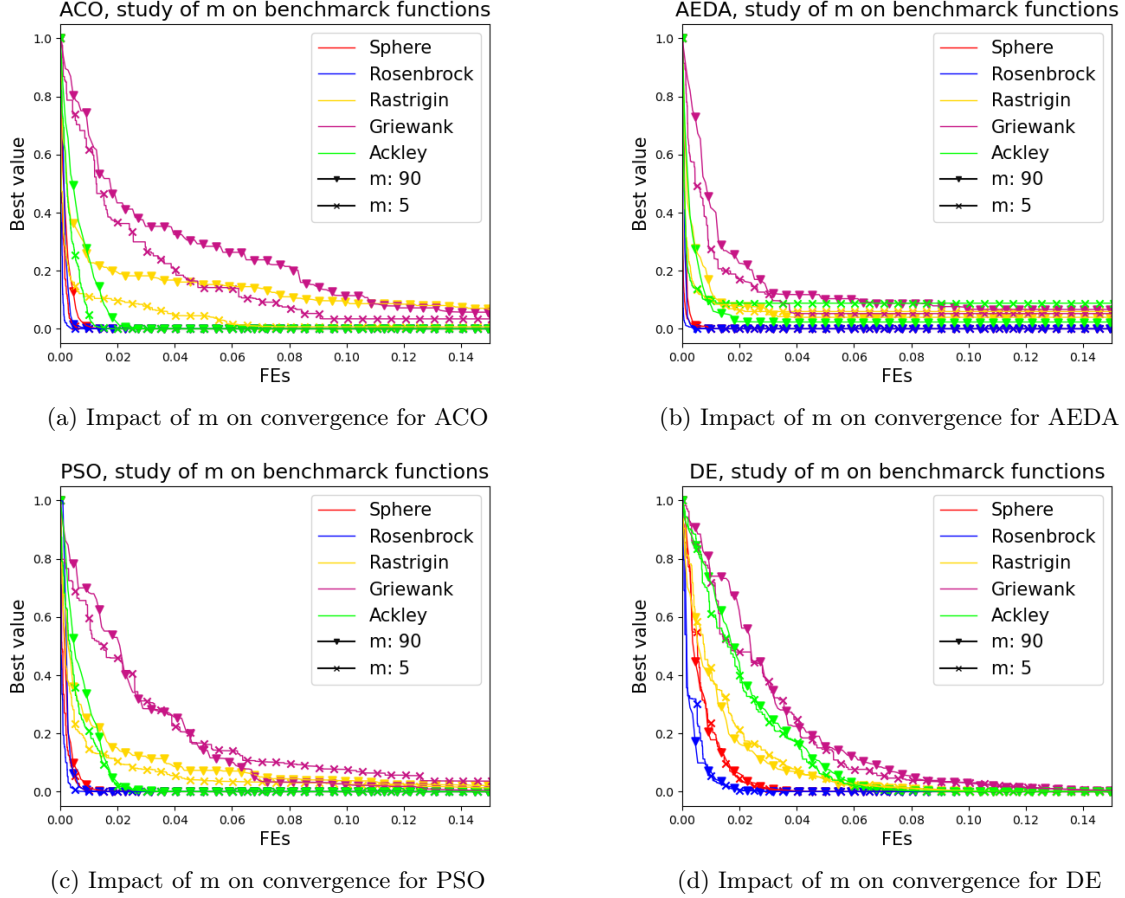


Figure 51: Impact of m on the early stage of the curve of convergence for ACO/AEDA/PSO/DE

We see on Figure 51a that increasing the update rate of the population, by decreasing m , increases the convergence rate for ACO. However, this observation doesn't hold for the rest of the algorithms. Indeed, on Figure 51b, 51c and 51d the curves of convergence for $m=5$ and $m=90$ are very similar.

We can recall that in the DE as well as in PSO the update of the population is done by replacing the solution in the solution archive only if the new created solution has a better fitness. Moreover, the selection of a solution in the solution archive is done by uniformly chose m solution among the k in the solution archive when $m < k$ and to chose the whole population when $m = k$. Thus, even if we improve faster the solutions in the solution archive, because there is no pressure towards selecting the fittest solution the impact is not so pronounced. As opposition to ACO which is working with a rank selection and a k -best update. However, AEDA also construct its new solutions using a rank selection and a k -best update but doesn't show an improve convergence rate by working with $m=5$.

		ACO		AEDA		DE		PSO	
		m=5	m=90	m=5	m=90	m=5	m=90	m=5	m=90
Sphere	min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Rosenbrock	min	2.18e-3	5.10e-1	1.63e-1	3.54e-3	3.58e-2	1.25e-2	5.85e-3	7.19e-1
	max	2.87e-3	6.64e-2	5.85e-1	8.95e-1	7.56e-1	1.35e-1	2.18e-1	1.48
Rastrigin	min	1.99	0.99	4.97	5.97	0.0	0.0	0.99	0.99
	max	2.98	6.96	20.89	6.96	0.0	0.0	2.984	1.99
Griewank	min	0.0	0.0	1.23e-2	4.43e-2	0.0	0.0	0.0	0.0
	max	2.46e-2	2.96e-2	1.08e-1	7.63e-2	0.0	0.0	1.48e-2	7.40e-3
Ackley	min	0.0	0.0	1.22	0.0	0.0	0.0	0.0	0.0
	max	0.0	0.0	1.22	0.0	0.0	0.0	0.0	0.0

Table 1: Minimum and maximum values reached by each metaheuristic on benchmark functions for m=5 and m=90 highlighting the best value found between each hyperparameter.

When looking at the final results for ACO we can see two interesting behaviours. First ACO performs better on the Rosenbrock function with m=5 than with m=90 both for the minimum and the maximum reached values. Secondly, on the Rastrigin function the range is from 1.99 to 2.98 with m=5 and from 0.99 to 6.96 with m=90. Thus, the best value is reached with m=90 but it also provides the worst value. Based on those observation the value of m=5 is chosen for ACO. For AEDA, we get the best results on the Rosenbrock and on the Ackley function with m=90. Then, on the Rastrigin and Griewank functions we get the best result with m=5 but the range of result is very large. We go from 4.97 up to 20.89 on Rastrigin and from 1.23e-2 to 1.08e-1 on Griewank, which is a huge difference considering the very low sensitivity of the Griewank function. Hence, the value of m=90 is chosen. In addition, choosing m=5 doesn't improve the convergence rate. About DE, as we get slightly better final results with m=90 and that there is no advantage, in terms of convergence, to chose m=5, then the value m=90 is chosen. With PSO, we get get a slightly better final result with m=5 on the Rosenbrock function and the convergence rate doesn't depends on m. Thus, the value m=5 is chosen.

Choice of the population size <k>

The hyperparameter k corresponds to the size of of the population. The higher k is, the more candidate solutions are present in the population. Thus, leading to more diversity.

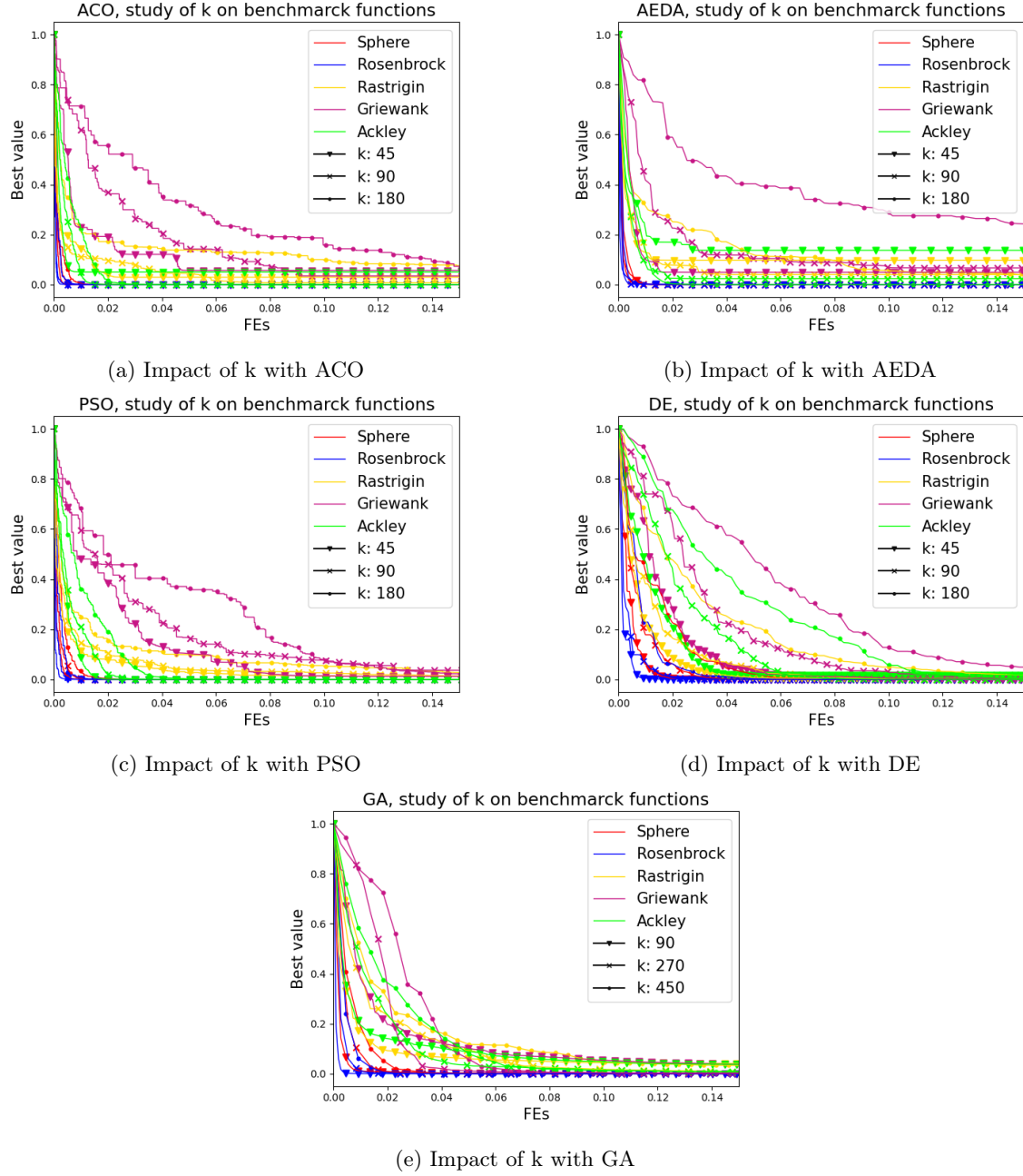


Figure 52: Impact of k on convergence for the five metaheuristics

We see that, the impact of the population size on the curve of convergence is similar on all the benchmark functions and for all the algorithms. Indeed, the increase of k is responsible of a slower convergence rate when using ACO, AEDA, PSO, DE and GA.

		Sphere		Rosenbrock		Rastrigin		Griewank		Ackley	
		min	max	min	max	min	max	min	max	min	max
ACO	k=45	0.0	0.0	2.78e-4	1.25e-2	0.99	15.92	2.21e-2	8.8563e-2	0.0	1.22
	k=90	0.0	0.0	2.18e-3	2.88e-3	1.99	2.98	0.0	2.4610e-2	0.0	0.0
	k=180	0.0	0.0	1.94e-2	3.05e-2	0.0	0.99	0.0	1.4780e-2	0.0	0.0
AEDA	k=45	0.0	0.0	4.05e-1	4.48	3.98	15.92	2.71e-2	3.94e-2	1.22	1.74
	k=90	0.0	0.0	3.54e-3	0.90	5.97	6.96	4.43e-2	7.63e-2	0.0	0.0
	k=180	0.0	0.0	2.30	2.65	0.99	4.97	7.40e-3	5.29e-2	0.0	0.0
DE	k=45	0.0	0.0	8.44e-2	8.18e-1	0.0	2.98	0.0	7.51e-3	0.0	0.0
	k=90	0.0	0.0	1.25e-2	1.35e-1	0.0	0.0	0.0	0.0	0.0	0.0
	k=180	0.0	0.0	1.21e-1	7.25e-1	0.0	0.0	0.0	0.0	0.0	0.0
PSO	k=45	0.0	0.0	4.21e-2	1.27	1.99	4.97	0.0	1.48e-2	0.0	0.0
	k=90	0.0	0.0	5.84e-3	0.22	0.99	2.98	0.0	1.48e-2	0.0	0.0
	k=180	0.0	0.0	3.74e-3	0.06	0.0	0.0	0.0	0.0	0.0	0.0
GA	k=90	0.0	0.0	6.33	8.42	2.98	6.96	0.0	5.67e-2	0.0	0.0
	k=270	0.0	0.0	8.05e-4	8.55	0.99	0.99	0.0	0.0	0.0	0.0
	k=450	0.0	0.0	4.3589e-2	0.16	1.99	2.98	0.0	0.0	0.0	0.0

Table 2: Minimum and maximum value for each metaheuristic on benchmark functions with $k=5*D$, $10*D$, $20*D$ for ACO, AEDA, DE and PSO and $k=10*D$, $30*D$ and $50*D$ for GA, where D is the number of decision variables and highlighting the best value found among each hyperparameter.

About ACO, if we look at the results using $k=45$, we get the best result the Rosenbrock function. Unfortunately, the results on the Griewank function are of bad quality. Indeed, $2.21e-2$ on the Griewank function correspond to the solution vector $(0, -4.4, 0, 0, 0, 0, 8.3, 0, 0)$ which is far from the optimum. Also, the range on the Rastrigin function is much higher with $k=45$ than with $k=90$ or $k=180$. The advantage of $k=90$ over $k=180$ is that it converges faster which result in better values on the Rosenbrock function. But, we have less diversity which allows less exploration. Hence, when using $k=90$ we get stuck on a local minimum for the Rastrigin function while using $k=180$ allow to find the global minimum. Therefore, $k=180$ is chosen. About AEDA, the value $k=180$ is chosen as it reaches the best result on Rastrigin and Griewank thanks to a higher diversity, even though the best result on the Rosenbrock function is with $k=90$. About DE, $k=90$ is chosen as it reaches the best, or similar, values on every benchmark function. Even though $k=45$ converge faster, it doesn't always find the optimum value on the Rastrigin and the Griewank functions. About PSO, we choose $k=180$ even though it is slower because it gives the best final results on all the benchmark functions. About GA, we choose $k=270$ even though it is slower because it gives the best final results on all the benchmark functions.

6.3.7 Comparison with tuned hyperparameters

Tuned ACO:

xi	q	m	k
0.6795	0.08	5	180

Tuned AEDA:

NP_{best}	m	k
$10(0.055\% of k)$	180	180

Tuned PSO :

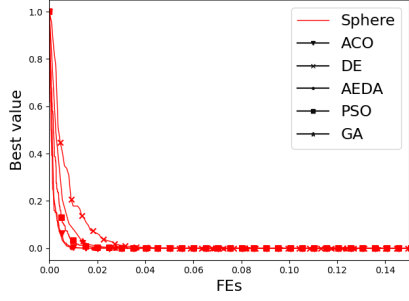
c	m	k
1.0	5	180

Tuned DE_rand_1 with binary crossover :

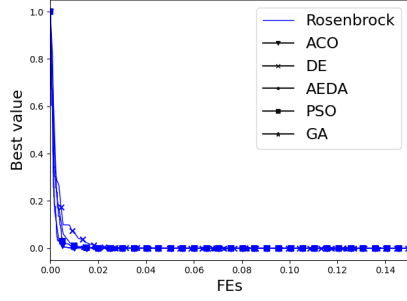
F	CR	m	k
0.1	0.2	90	90

Tuned GA with Heuristic crossover and Power mutation:

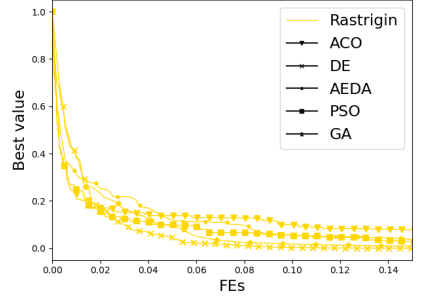
cxp	mp	StrengthPM	tr_size	elite_size	m	k
0.9	0.5	0.25	3	10	270	270



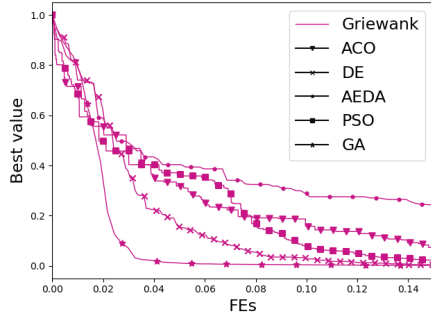
(a) Comparison of metaheuristic on the Sphere function



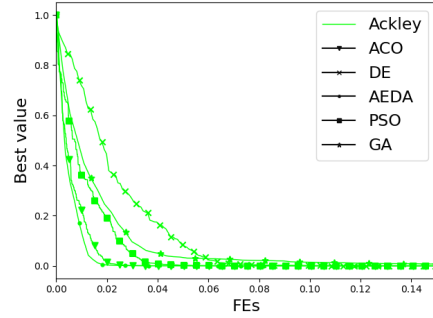
(b) Comparison of metaheuristic on the Rosenbrock function



(c) Comparison of metaheuristic on the Rastrigin function



(d) Comparison of metaheuristic on the Griewank function



(e) Comparison of metaheuristic on the Ackley function

Figure 53: Comparison of the five metaheuristics on all the benchmark functions

First, we see that the DE algorithm converges slowly the Sphere [53a](#) and the Rosenbrock [53b](#) functions. It also converges more slowly than the other metaheuristic algorithms on the Ackley function. We can also notice that, on this Ackley function, the fastest algorithms are ACO and AEDA. Finally, on the Griewank function the GA algorithm converges quite fast and the AEDA quite slowly compared to the others. For the rest, there are no significant differences among the five studied algorithms in terms of convergence.

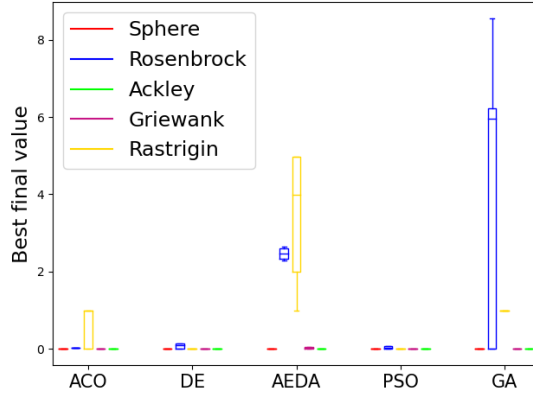


Figure 54: Boxplot for different metaheuristic on benchmark functions over 5 runs

In terms of final results, we can notice the particularly good performance of ACO, DE and PSO algorithms. AEDA seems to have more trouble when dealing with complex function such as Rosenbrock, Rastrigin and Griewank. About GA, it is particularly bad we trying to find the optimal solution on the Rosenbrock function.

6.3.8 Discussion and remarks

In the paper [6] they also worked on the Sphere, the Rosenbrock, the Rastrigin, the Griewank and the Ackley function. But, they used a maximum number of function evaluation of $1e6$ instead of $1e5$, and 10 instead of 9 decision variable. However, the obtained the following results :

- Sphere : find zero at each run
- Rosenbrock : find values in the interval $[0, 37.4]$
- Rastrigin : find values in the interval $[2.93, 24.6]$
- Griewank : find values in the interval $[0, 4.67e-2]$
- Ackley : find zero at each run

We see that the implemented algorithms work in a similar fashion. They struggle in general more for the Rosenbrock, the Rastrigin and the Griewank functions. Additionally, each of the the tuned algorithm implemented finds better results. It may be because of a more adapted tuning and/or a lower dimension of the problem.

ACO. This study have found the same optimal value for they hyperparameters ξ and m as in the paper [6] from which it was inspired. However, we also found a bit higher value for q and the chosen value for the population size is 180 instead of 90.

AEDA. The results of the tuning have found smaller fraction of the population to create new solutions than in [8] but this smaller fraction is accompanied with a higher population size. This allow to still use enough information in order to create a new solutions.

PSO. The acceleration coefficient, c , is chosen to 1.0 instead of 2.0 ([7]) has it has showed better result. Additionally, the size of the population also increased. The hyperparameter c multiplies the difference between 2 solutions in the solution archive and, the larger is the size of the solution archive, the more it is diverse. Hence, an hypothesis to explain the differences of value from the tuning and the referenced value is that, decreasing c and increasing the difference between two solutions or increasing c and reducing the difference between two solutions is similar. And in our study the first option have showed slightly better results.

DE. About the hyperparameters of DE, the only difference is to chose $F=0.1$ instead of $F=0.2$ ([5]). We see on the Figure 48 that the variation is very small.

GA. The optimal value found for the crossover and mutation probability is higher than in the reference papers [11], [19]. However, the chosen values are similar to the ones used in many Genetic Algorithms [18], [41], [21] or [42]. The fact that the elitism size is also higher can be understand by the fact that we have a higher crossover and mutation probability. Indeed, with the elitism we force some solutions to be passed from one iteration to another, but as the crossover and the mutation operators are only applied with a certain probability, then some solutions, independently to the elitism strategy, will be unchanged. Because the chosen mutation and crossover probability are higher than in the reference papers less solution will be unchanged, which is countered by a higher elitism size. Finally, a population size of 30 times the number of decision variables is chosen instead of 10 times. This difference can be due to the fact that many papers, including the reference paper, do not tune this hyperparameter.

6.4 Results on Real case

6.4.1 Choice of algorithm

Here under is the curve of convergence of the different algorithms applied to the hydrogen supply chain design (HSCD).

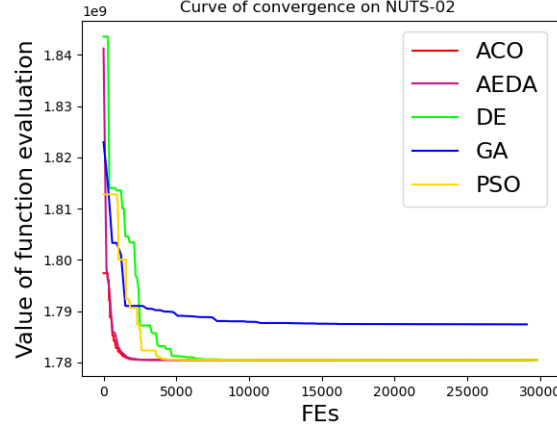
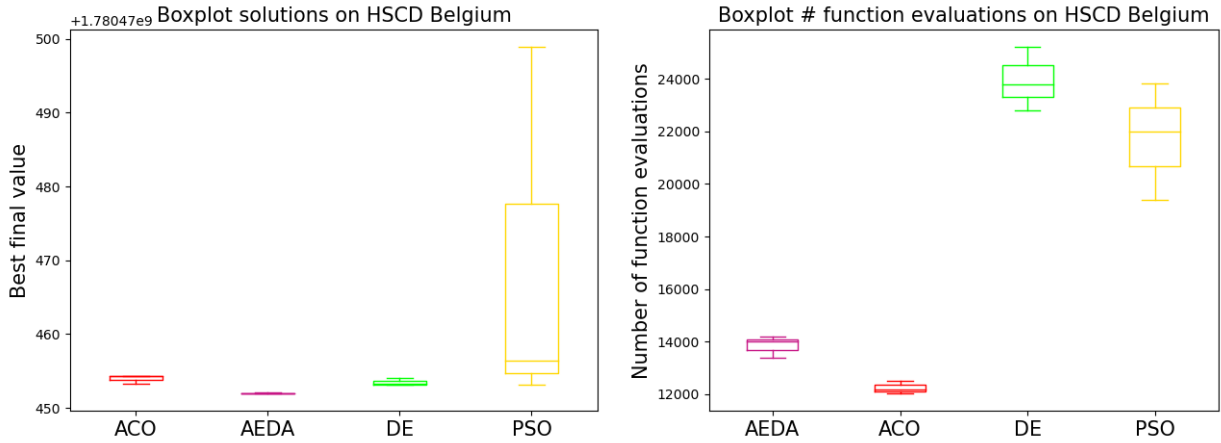


Figure 55: Curve of convergence of ACO, AEDA, DE, PSO and GA on the HSCD

Based on the Figure 55 we can already identify that, the Genetic Algorithm (with the tuned hyperparameters) is not performing well on this optimization problem. Hence, it will not be part of the further investigations.

For the other algorithms, we can identify a fast convergence rate for ACO and AEDA in comparison to DE and PSO. To further examine the speed of convergence and the solution found by each algorithm, five other runs are performed. These runs use a maximum number of stagnation iterations and the restart strategy (see section 2). The following boxplots show the best solution found by each algorithm on the 5 runs and the total number of iterations required.



(a) Boxplot of the final results for ACO, AEDA, DE and (b) Boxplot of the number of iteration for ACO, AEDA, DE and PSO on the HSCD

We see that ACO, AEDA and DE find similar solution but DE takes much more function evaluations to achieve this result. Therefore, ACO and AEDA as considered as the best choice of metaheuristic algorithm for solving the optimization problem of HSCD describe in [4](#) with in the accent more put on AEDA is it reaches the best solution among the studied algorithms.

6.4.2 Impact of discretization

Impact on convergence/final result for AEDA

A discretization level of 2 means that the decision variable can take two other values than the minimum value. Hence, with a discretization level of 2 the possible values are : min, mean, max.

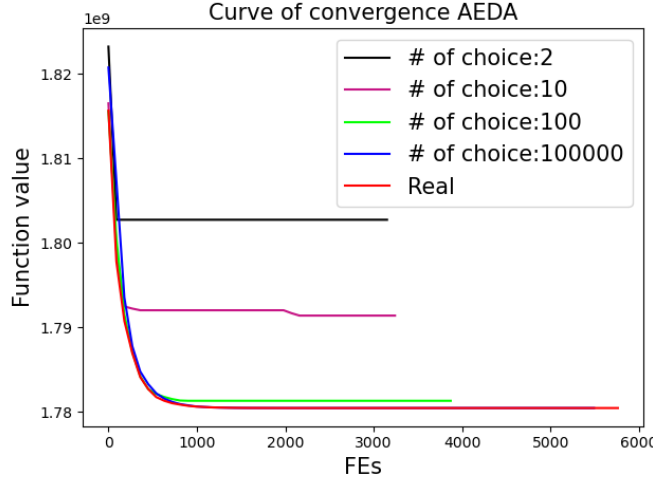
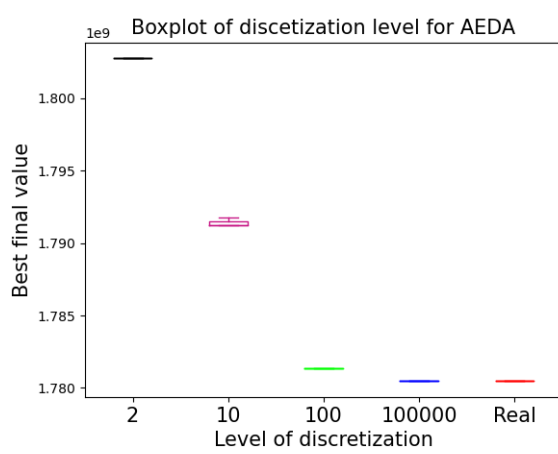
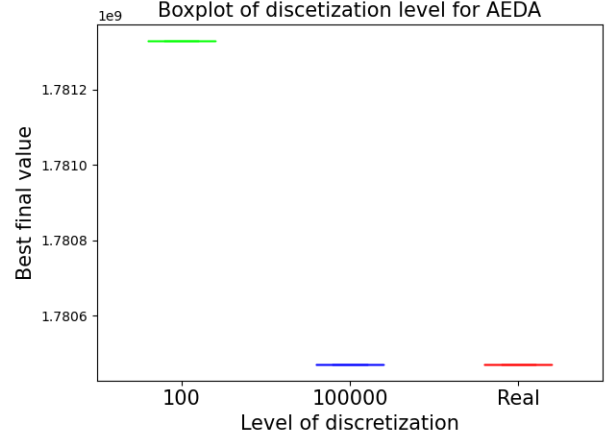


Figure 57: Curve of convergence depending on the discretization level with AEDA

A first observation is that when using a discretization such that there is 100 000 possible values for each decision variables we get the same convergence curve as when solving the optimization problem using real decision variables. This shows that the implemented way of dealing with discrete variables works. Then the more we decrease the number of possible values the more the algorithm converge fast but also the more we decrease the quality of the solution. To further investigate this here under is the boxplot of the values found for each discretization level and for real variables. And, the boxplot of the number of function evaluation before stagnation.



(a) Boxplot of the final values depending on the discretization level with AEDA



(b) Boxplot of the final values depending on the discretization level with AEDA zoomed

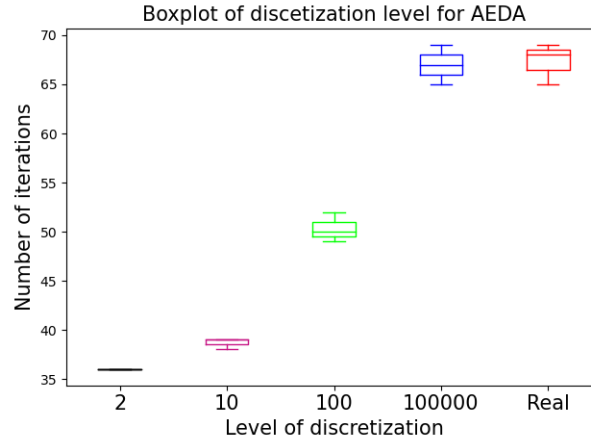


Figure 59: Boxplot of the number of iterations before stagnation depending on the discretization level with AEDA

We see that the advantages in terms of function evaluation is null when considering a discretization level of 100 000. For smaller discretization level their is an interest in term of function evaluation, and thus of computation time, of working with discrete variables. But, this advantage isn't worth the bad quality of the found solution.

Impact on convergence/final result for ACO

We see on the following figures that we reach the same conclusion for ACO as for AEDA in terms of impact of the discretization of decision variables. We see that the advantages in terms of function evaluation is null when considering a discretization level of 100 000. For smaller discretization level their is an interest in term of function evaluation, and thus of computation time, of working with discrete variables. But, this advantage isn't worth the bad quality of the found solution

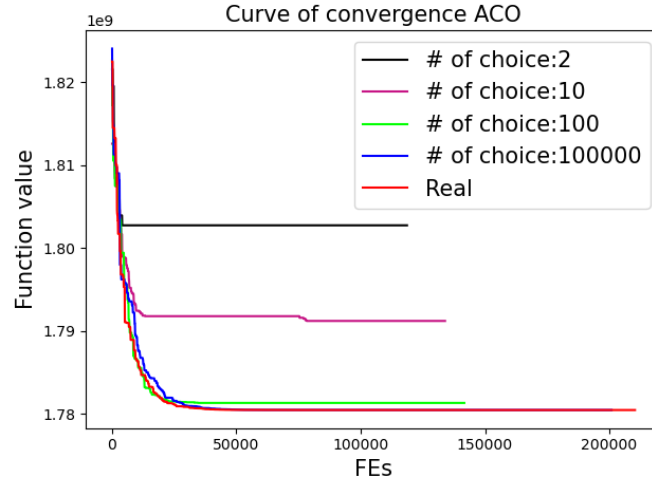
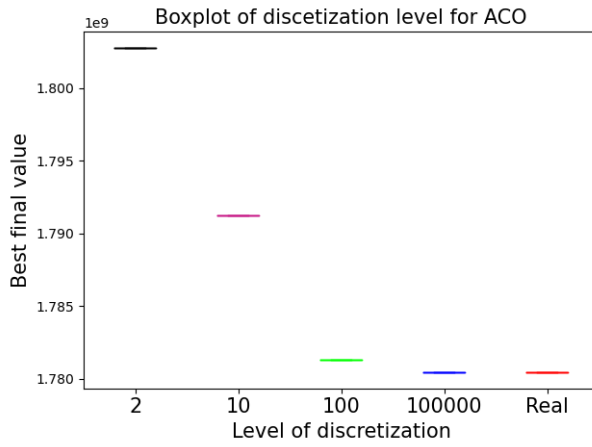
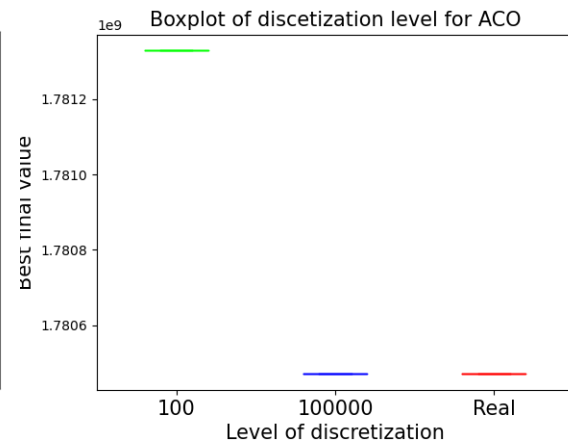


Figure 60: Curve of convergence depending on the discretization level with ACO



(a) Boxplot of the final values depending on the discretization level with ACO



(b) Boxplot of the final values depending on the discretization level with ACO with zoom on level 100, 100 000 and Real

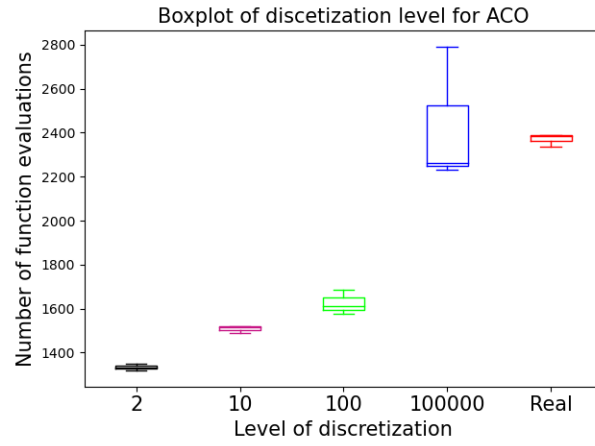


Figure 62: Boxplot of the number of iterations before stagnation depending on the discretization level with ACO

7 Conclusion

The purpose of this master thesis was to identify the best metaheuristic algorithm to solve the problem of hydrogen supply chain design (HSCD) including production and transport technologies in Belgium. The formulation used of hydrogen supply chain design (HSCD) keeps into consideration the non-linear constraints is based on Mixed-Integer Non-Linear Programming (MINLP). The strategy retained to solve the HSCD is to hybridize a metaheuristic algorithm with a linear solver with the metaheuristic handling the non-linearities of the problem. To do so, we have followed a four steps approach.

First, a review of existing Genetic Algorithms was performed in order to extend and deepen the comparison among Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Differential Evolution (DE) and Adaptive Estimation of Distribution Algorithm (AEDA) done in [5]. This work has identified the Laplace and the Heuristic crossover as well as the Power and the non-uniform mutation as possible implementations of a multi-variate GA. We have implemented those two types of crossovers and mutations and compared the performances on five benchmark functions. This has allowed us to conclude that the Heuristic-Power GA was the best performing algorithm and it is thus recommended as version of GA for future work dealing with both continuous and discrete variable. Additionally, this review identified possible variants of DE namely DE/rand/1, DE/best/1 and DE/best/2 as well as a binary and exponential crossover. Our analysis identified DE/rand/1/bin as the best variant of DE.

Once selected the best variant for each metaheuristic, the second step consisted in defining the best hyperparameters each of them. We thus performed an extended review of ACO, PSO, DE and AEDA to understand their behaviours, specificities and to identify the hyperparameters present in each of them. Two hyperparameters are common to all the metaheuristics, namely the population size (k) and the number of new solutions created per iteration (m). On top of these common hyperparameters, in ACO, the user has to chose a value to influence the variance (xi) and one for the rank selection (q). In AEDA, the number of best solutions used when creating new solutions (NP_{best}) needs to be defined while in PSO, the acceleration coefficient (c) is an additional hyperparameter. In our version of GA (HX-PM), we need the crossover probability (cxp), the mutation probability (mp), the strength of the power mutation, the tournament size and the elite size. Finally, DE has to chose the scaling factor (F) and the crossover rate (CR).

We performed the tuning of the five retained metaheuristics to have the best results performance of metaheuristic on benchmark functions of dimension 9. We compared them according to the same performance criteria as above. We obtained the best results for ACO with $xi = 0.6795$, $q = 0.08$, $m = 5$ and $k=180$; for AEDA with $NP_{best}=10$, $m=180$ and $k=180$; for PSO with $c=1.0$, $m=5$ and $k=180$, for DE with DE/rand/1/bin, $F=0.1$, $CR=0.2$ $m=90$ and $k=90$; for HX-PM GA with $cxp=0.9$, $mp=0.5$, tournament size=3, elite size=10, $m = 270$ and $k=270$. These values are thus recommended when using these metaheuristic algorithms.

In a third step, we compared the five tuned metaheuristics with the same performance criteria. This comparison was first done on the five benchmark functions and showed particularly good results using ACO, DE and PSO. We then performed this comparison on our target problem of HSCD in Belgium. This led to the conclusion that the best performance was obtained using AEDA and ACO. Good performance are also obtained using PSO and DE, while GA did not perform well.

Finally, a study of the impact of the discretization was performed on the best performing algorithms: ACO and AEDA. It was showed that in the case of HSCD the discretization of the variable with 3, 10 and 100 choices per variable converged faster than with real variables, but gave rise to bad results. While going to 100 000 choices provided results as good as with real variable but didn't show any interest in terms of number of convergence.

In conclusion, it is recommended for future work on the hydrogen supply chain to use the Adaptive estimation of distribution algorithm with the hyperparameters mentioned above and without discretization of the variable.

8 Perspectives

This work can be further improved by proposing a deeper analysis of the metaheuristic algorithms on the design of hydrogen supply chain. One can perform a tuning of the metaheuristics on this specific real-world problem and see if there are differences with the tuning performed on the benchmark functions. Similarly, more values for the discretization could be tested. It would also be interesting to test these metaheuristics to a larger scale than Belgium which will induce a higher number of decision variables.

One could also increase the precision of the tuning strategy by testing more values and more combination of hyperparameters. Nonetheless, this would need to be rigorous and would require a lot of time. Another possibility might be to use a machine learning algorithm to tune the metaheuristic algorithm (e.g. using a grid search technique).

Another aspect that could lead to a better resolution method would be to see if it is possible to implement an adaptive estimation of distribution algorithm which takes into consideration the dependency among the variables while using the same encoding scheme as in this work. Some of the decision variables in the hydrogen supply chain, like the input flow of water and electricity, can be directly computed based on the output flow of hydrogen. But there are also some dependencies among the output flow of the different production units which are not taken into consideration in this study.

Finally, one could investigate the possibility of using more adaptive strategy (like the one used in AEDA to tune w during the search process) inside the metaheuristic algorithms in order to reduce the number of hyperparameters.

Bibliography

- [1] Hydrogen Council. *Path to hydrogen competitiveness - A cost perspective*. Jan. 2020.
- [2] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. en. Cambridge, MA, USA: MIT Press, Mar. 2019. ISBN: 978-0-262-03942-0.
- [3] Sergio Nesmachnow. “An overview of metaheuristics: Accurate and efficient methods for optimisation”. In: *International Journal of Metaheuristics* 3 (Dec. 2014), p. 320. DOI: [10.1504/IJMHEUR.2014.068914](https://doi.org/10.1504/IJMHEUR.2014.068914).
- [4] D.H. Wolpert and W.G. Macready. “No free lunch theorems for optimization”. In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997). Conference Name: IEEE Transactions on Evolutionary Computation, pp. 67–82. ISSN: 1941-0026. DOI: [10.1109/4235.585893](https://doi.org/10.1109/4235.585893).
- [5] Feng Wang, Heng Zhang, and Aimin Zhou. “A particle swarm optimization algorithm for mixed-variable optimization problems”. en. In: *Swarm and Evolutionary Computation* 60 (Feb. 2021), p. 100808. ISSN: 2210-6502. DOI: [10.1016/j.swevo.2020.100808](https://doi.org/10.1016/j.swevo.2020.100808). URL: <https://www.sciencedirect.com/science/article/pii/S2210650220304612> (visited on 04/30/2022).
- [6] Tianjun Liao et al. “Ant Colony Optimization for Mixed-Variable Optimization Problems”. In: *IEEE Transactions on Evolutionary Computation* 18.4 (Aug. 2014). Conference Name: IEEE Transactions on Evolutionary Computation, pp. 503–518. ISSN: 1941-0026. DOI: [10.1109/TEVC.2013.2281531](https://doi.org/10.1109/TEVC.2013.2281531).
- [7] Ying Lin et al. “A hybrid differential evolution algorithm for mixed-variable optimization problems”. en. In: *Information Sciences* 466 (Oct. 2018), pp. 170–188. ISSN: 0020-0255. DOI: [10.1016/j.ins.2018.07.035](https://doi.org/10.1016/j.ins.2018.07.035). URL: <https://www.sciencedirect.com/science/article/pii/S0020025516318163> (visited on 04/30/2022).
- [8] Wen Shi et al. “An Adaptive Estimation of Distribution Algorithm for Multipolicy Insurance Investment Planning”. In: *IEEE Transactions on Evolutionary Computation* 23.1 (Feb. 2019). Conference Name: IEEE Transactions on Evolutionary Computation, pp. 1–14. ISSN: 1941-0026. DOI: [10.1109/TEVC.2017.2782571](https://doi.org/10.1109/TEVC.2017.2782571).
- [9] Kusum Deep et al. “A real coded genetic algorithm for solving integer and mixed integer optimization problems”. In: *Applied Mathematics and Computation* 212 (June 2009), pp. 505–518. DOI: [10.1016/j.amc.2009.02.044](https://doi.org/10.1016/j.amc.2009.02.044).
- [10] Xin-She Yang. “Nature-inspired optimization algorithms: Challenges and open problems”. en. In: *Journal of Computational Science* 46 (Oct. 2020), p. 101104. ISSN: 18777503. DOI: [10.1016/j.jocs.2020.101104](https://doi.org/10.1016/j.jocs.2020.101104). URL: <https://linkinghub.elsevier.com/retrieve/pii/S1877750320300144> (visited on 04/30/2022).
- [11] Kusum Deep and Manoj Thakur. “A new crossover operator for real coded genetic algorithms”. In: *Applied Mathematics and Computation* 188 (May 2007), pp. 895–911. DOI: [10.1016/j.amc.2006.10.047](https://doi.org/10.1016/j.amc.2006.10.047).
- [12] Uday K. Chakraborty and Cezary Z. Janikow. “An analysis of Gray versus binary encoding in genetic search”. en. In: *Information Sciences*. Evolutionary Computation 156.3 (Nov. 2003), pp. 253–269. ISSN: 0020-0255. DOI: [10.1016/S0020-0255\(03\)00178-6](https://doi.org/10.1016/S0020-0255(03)00178-6). URL: <https://www.sciencedirect.com/science/article/pii/S0020025503001786> (visited on 04/30/2022).
- [13] Kumara Sastry, David Goldberg, and Graham Kendall. “Genetic Algorithms”. en. In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. by Edmund K. Burke and Graham Kendall. Boston, MA: Springer US, 2005, pp. 97–125. ISBN: 978-0-387-28356-2. DOI: [10.1007/0-387-28356-0_4](https://doi.org/10.1007/0-387-28356-0_4). URL: https://doi.org/10.1007/0-387-28356-0_4 (visited on 04/30/2022).
- [14] Said Bourazza. “Variantes d’algorithmes génétiques appliquées aux problèmes d’ordonnancement”. In: (2006), p. 188.
- [15] J Holland. “Adaptation in Nature and Artificial Systems. University of Michigan Press, Ann Arbor, MI, USA.” In: (1975).
- [16] D.E Goldberg. “Genetic Algorithms in Search, Optimization and Machine Learning. AddisonWesley, Reading, MA, USA”. In: (1989).
- [17] David Goldberg Graham Kendall Kumara Sastry. “Chapter 27 Genetic algorithms and other global search strategies”. en. In: *Data Handling in Science and Technology*. Ed. by D. L. Massart et al.

- Vol. 20. Handbook of Chemometrics and Qualimetrics: Part A. Elsevier, Jan. 1998, pp. 805–848. DOI: [10.1016/S0922-3487\(97\)80057-3](https://doi.org/10.1016/S0922-3487(97)80057-3). URL: <https://www.sciencedirect.com/science/article/pii/S0922348797800573> (visited on 04/30/2022).
- [18] Abid Hussain et al. “Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator”. en. In: *Computational Intelligence and Neuroscience* 2017 (Oct. 2017). Publisher: Hindawi, e7430125. ISSN: 1687-5265. DOI: [10.1155/2017/7430125](https://doi.org/10.1155/2017/7430125). URL: <https://www.hindawi.com/journals/cin/2017/7430125/> (visited on 04/30/2022).
- [19] Kusum Deep and Manoj Thakur. “A new mutation operator for real coded genetic algorithms”. en. In: *Applied Mathematics and Computation* 193.1 (Oct. 2007), pp. 211–230. ISSN: 0096-3003. DOI: [10.1016/j.amc.2007.03.046](https://doi.org/10.1016/j.amc.2007.03.046). URL: <https://www.sciencedirect.com/science/article/pii/S0096300307003918> (visited on 04/30/2022).
- [20] Alden Wright. “Genetic Algorithms for Real Parameter Optimization”. In: *Foundations of Genetic Algorithms* 1 (June 1999). ISSN: 9780080506845. DOI: [10.1016/B978-0-08-050684-5.50016-1](https://doi.org/10.1016/B978-0-08-050684-5.50016-1).
- [21] Noraini Razali and John Geraghty. “Genetic Algorithm Performance with Different Selection Strategies in Solving TSP”. In: vol. 2. Jan. 2011.
- [22] Rainer Storn and Kenneth Price. “Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces”. en. In: *Journal of Global Optimization* 11.4 (Dec. 1997), pp. 341–359. ISSN: 1573-2916. DOI: [10.1023/A:1008202821328](https://doi.org/10.1023/A:1008202821328). URL: <https://doi.org/10.1023/A:1008202821328> (visited on 04/30/2022).
- [23] Chuan Lin, Anyong Qing, and Quanyuan Feng. “A comparative study of crossover in differential evolution”. In: *J. Heuristics* 17 (Dec. 2011), pp. 675–703. DOI: [10.1007/s10732-010-9151-1](https://doi.org/10.1007/s10732-010-9151-1).
- [24] Karol R. Opara and Jarosław Arabas. “Differential Evolution: A survey of theoretical analyses”. en. In: *Swarm and Evolutionary Computation* 44 (Feb. 2019), pp. 546–558. ISSN: 2210-6502. DOI: [10.1016/j.swevo.2018.06.010](https://doi.org/10.1016/j.swevo.2018.06.010). URL: <https://www.sciencedirect.com/science/article/pii/S2210650217304224> (visited on 04/30/2022).
- [25] J. Kennedy and R. Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95 - International Conference on Neural Networks*. Vol. 4. Nov. 1995, 1942–1948 vol.4. DOI: [10.1109/ICNN.1995.488968](https://doi.org/10.1109/ICNN.1995.488968).
- [26] Y. Shi and R. Eberhart. “A modified particle swarm optimizer”. In: *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*. May 1998, pp. 69–73. DOI: [10.1109/ICEC.1998.699146](https://doi.org/10.1109/ICEC.1998.699146).
- [27] Yuhui Shi. “Particle Swarm Optimization”. en. In: (2004), p. 6.
- [28] Jing Liang et al. “Comprehensive Learning Particle Swarm Optimiser for Global Optimisation of Multimodal Functions”. In: *Evolutionary Computation, IEEE Transactions on* 10 (July 2006), pp. 281–295. DOI: [10.1109/TEVC.2005.857610](https://doi.org/10.1109/TEVC.2005.857610).
- [29] Rui Mendes, James Kennedy, and João Neves. “The Fully Informed Particle Swarm: Simpler, Mabe Better”. In: *Evolutionary Computation, IEEE Transactions on* 8 (July 2004), pp. 204–210. DOI: [10.1109/TEVC.2004.826074](https://doi.org/10.1109/TEVC.2004.826074).
- [30] R. Eberhart and Yuhui Shi. “Particle swarm optimization: developments, applications and resources”. In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)* (2001). DOI: [10.1109/CEC.2001.934374](https://doi.org/10.1109/CEC.2001.934374).
- [31] M Dorigo. “Optimization, Learning, and Natural Algorithms. Ph.D. Thesis. Politecnico di Milano”. en. In: (1992).
- [32] Krzysztof Socha. “Ant Colony Optimization for Continuous and Mixed-Variable Domains”. en. In: (), p. 193.
- [33] Rubén Armañanzas et al. “A review of estimation of distribution algorithms in bioinformatics”. en. In: *BioData Mining* 1.1 (Dec. 2008), p. 6. ISSN: 1756-0381. DOI: [10.1186/1756-0381-1-6](https://doi.org/10.1186/1756-0381-1-6). URL: <https://biodatamining.biomedcentral.com/articles/10.1186/1756-0381-1-6> (visited on 05/31/2022).
- [34] Jianyong Sun, Qingfu Zhang, and Edward P. K. Tsang. “DE/EDA: A new evolutionary algorithm for global optimization”. In: *Information Sciences* (), pp. 249–262.

- [35] Xin-She Yang, Xing-Shi He, and Qin-Wei Fan. “Chapter 7 - Mathematical framework for algorithm analysis”. en. In: *Nature-Inspired Computation and Swarm Intelligence*. Ed. by Xin-She Yang. Academic Press, Jan. 2020, pp. 89–108. ISBN: 978-0-12-819714-1. DOI: [10.1016/B978-0-12-819714-1.00017-8](https://doi.org/10.1016/B978-0-12-819714-1.00017-8). URL: <https://www.sciencedirect.com/science/article/pii/B9780128197141000178> (visited on 04/30/2022).
- [36] Michael Hellwig and Hans-Georg Beyer. “Benchmarking evolutionary algorithms for single objective real-valued constrained optimization – A critical review”. en. In: *Swarm and Evolutionary Computation* 44 (Feb. 2019), pp. 927–944. ISSN: 2210-6502. DOI: [10.1016/j.swevo.2018.10.002](https://doi.org/10.1016/j.swevo.2018.10.002). URL: <https://www.sciencedirect.com/science/article/pii/S2210650218305406> (visited on 04/30/2022).
- [37] Olaf Mersmann, Mike Preuss, and Heike Trautmann. “Benchmarking Evolutionary Algorithms: Towards Exploratory Landscape Analysis”. en. In: *Parallel Problem Solving from Nature, PPSN XI*. Ed. by Robert Schaefer et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 73–82. ISBN: 978-3-642-15844-5. DOI: [10.1007/978-3-642-15844-5_8](https://doi.org/10.1007/978-3-642-15844-5_8).
- [38] Marie Demers Ph. D Jacques Roy Ph. D. *La filière de l’hydrogène : un avantage stratégique pour le Québec*. 2019.
- [39] *Green hydrogen in Europe- A regional assessment: Substituting existing production with electrolysis powered by renewables*. en. DOI: [10.1016/j.enconman.2020.113649](https://doi.org/10.1016/j.enconman.2020.113649). (Visited on 06/04/2022).
- [40] Avni Rexhepi, Adnan Maxhuni, and Agni Dika. *Analysis of the impact of parameters values on the Genetic Algorithm for TSP*.
- [41] Swati Swayamsiddha. “Chapter 4 - Bio-inspired algorithms: principles, implementation, and applications to wireless communication”. en. In: *Nature-Inspired Computation and Swarm Intelligence*. Ed. by Xin-She Yang. Academic Press, Jan. 2020, pp. 49–63. ISBN: 978-0-12-819714-1. DOI: [10.1016/B978-0-12-819714-1.00013-0](https://doi.org/10.1016/B978-0-12-819714-1.00013-0). URL: <https://www.sciencedirect.com/science/article/pii/B9780128197141000130> (visited on 04/30/2022).
- [42] Walchand College of Engineering et al. “CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW”. en. In: *ICTACT Journal on Soft Computing* 06.01 (Oct. 2015), pp. 1083–1092. ISSN: 09766561, 22296956. DOI: [10.21917/ijsc.2015.0150](https://doi.org/10.21917/ijsc.2015.0150). (Visited on 06/04/2022).
- [43] Victor H. Cantú, Catherine Azzaro-Pantel, and Antonin Ponsich. “A Novel Matheuristic based on bi-level optimization for the multi-Objective design of hydrogen supply chains”. en. In: *Computers & Chemical Engineering* 152 (Sept. 2021), p. 107370. ISSN: 0098-1354. DOI: [10.1016/j.compchemeng.2021.107370](https://doi.org/10.1016/j.compchemeng.2021.107370). URL: <https://www.sciencedirect.com/science/article/pii/S0098135421001484> (visited on 04/30/2022).
- [44] Rami S. El-Emam and Hasan Özcan. “Comprehensive review on the techno-economics of sustainable large-scale clean hydrogen production”. en. In: *Journal of Cleaner Production* 220 (May 2019), pp. 593–609. ISSN: 0959-6526. DOI: [10.1016/j.jclepro.2019.01.309](https://doi.org/10.1016/j.jclepro.2019.01.309). URL: <https://www.sciencedirect.com/science/article/pii/S0959652619303361> (visited on 04/30/2022).
- [45] Paolo Gabrielli et al. “Enabling low-carbon hydrogen supply chains through use of biomass and carbon capture and storage: A Swiss case study”. en. In: *Applied Energy* 275 (Oct. 2020), p. 115245. ISSN: 0306-2619. DOI: [10.1016/j.apenergy.2020.115245](https://doi.org/10.1016/j.apenergy.2020.115245). URL: <https://www.sciencedirect.com/science/article/pii/S0306261920307571> (visited on 04/30/2022).
- [46] Emma S. Hanley, JP Deane, and BP Ó Gallachóir. “The role of hydrogen in low carbon energy futures—A review of existing perspectives”. en. In: *Renewable and Sustainable Energy Reviews* 82 (Feb. 2018), pp. 3027–3045. ISSN: 1364-0321. DOI: [10.1016/j.rser.2017.10.034](https://doi.org/10.1016/j.rser.2017.10.034). URL: <https://www.sciencedirect.com/science/article/pii/S1364032117314089> (visited on 04/30/2022).
- [47] *A multi-objective multi-population ant colony optimization for economic emission dispatch considering power system security*. en. DOI: [10.1016/j.apm.2017.01.001](https://doi.org/10.1016/j.apm.2017.01.001). (Visited on 06/03/2022).
- [48] Jesus Ochoa Robles, Catherine Azzaro-Pantel, and Alberto Aguilar-Lasserre. “Optimization of a hydrogen supply chain network design under demand uncertainty by multi-objective genetic algorithms”. en. In: *Computers & Chemical Engineering* 140 (Sept. 2020), p. 106853. ISSN: 0098-1354. DOI: [10.1016/j.compchemeng.2020.106853](https://doi.org/10.1016/j.compchemeng.2020.106853). URL: <https://www.sciencedirect.com/science/article/pii/S0098135419307161> (visited on 04/30/2022).

- [49] Young-Bin Woo and Byung Soo Kim. “A genetic algorithm-based matheuristic for hydrogen supply chain network problem with two transportation modes and replenishment cycles”. en. In: *Computers & Industrial Engineering* 127 (Jan. 2019), pp. 981–997. ISSN: 0360-8352. DOI: [10.1016/j.cie.2018.11.027](https://doi.org/10.1016/j.cie.2018.11.027). URL: <https://www.sciencedirect.com/science/article/pii/S0360835218305679> (visited on 04/30/2022).
- [50] Yonglin Ren and Anjali Awasthi. “Investigating Metaheuristics Applications for Capacitated Location Allocation Problem on Logistics Networks”. In: *Studies in Computational Intelligence* 581 (Dec. 2015), pp. 213–238. ISSN: 978-3-319-13131-3. DOI: [10.1007/978-3-319-13132-0_9](https://doi.org/10.1007/978-3-319-13132-0_9).
- [51] Reza Zanjirani Farahani and Masoud Hekmatfar. *Facility Location: Concepts, Models, Algorithms and Case Studies*. en. Google-Books-ID: a7JhwTwHveMC. Springer Science & Business Media, July 2009. ISBN: 978-3-7908-2151-2.
- [52] Tarek El-mihoub et al. “Hybrid Genetic Algorithms: A Review”. In: *Engineering Letters* 3(2) (Aug. 2006).
- [53] Xin-She Yang. “Chapter 5 - Genetic Algorithms”. en. In: *Nature-Inspired Optimization Algorithms*. Ed. by Xin-She Yang. Oxford: Elsevier, Jan. 2014, pp. 77–87. ISBN: 978-0-12-416743-8. DOI: [10.1016/B978-0-12-416743-8.00005-1](https://doi.org/10.1016/B978-0-12-416743-8.00005-1). URL: <https://www.sciencedirect.com/science/article/pii/B9780124167438000051> (visited on 04/30/2022).
- [54] Gilbert Syswerda. “A Study of Reproduction in Generational and Steady-State Genetic Algorithms”. en. In: *Foundations of Genetic Algorithms*. Ed. by GREGORY J. E. Rawlins. Vol. 1. Elsevier, Jan. 1991, pp. 94–101. DOI: [10.1016/B978-0-08-050684-5.50009-4](https://doi.org/10.1016/B978-0-08-050684-5.50009-4). URL: <https://www.sciencedirect.com/science/article/pii/B9780080506845500094> (visited on 04/30/2022).
- [55] Randy L. Haupt and S. E. Haupt. *Practical genetic algorithms*. en. 2nd ed. Hoboken, N.J: John Wiley, 2004. ISBN: 978-0-471-45565-3.
- [56] Xin-She Yang and Mehmet Karamanoglu. “Chapter 1 - Nature-inspired computation and swarm intelligence: a state-of-the-art overview”. en. In: *Nature-Inspired Computation and Swarm Intelligence*. Ed. by Xin-She Yang. Academic Press, Jan. 2020, pp. 3–18. ISBN: 978-0-12-819714-1. DOI: [10.1016/B978-0-12-819714-1.00010-5](https://doi.org/10.1016/B978-0-12-819714-1.00010-5). URL: <https://www.sciencedirect.com/science/article/pii/B9780128197141000105> (visited on 04/30/2022).
- [57] Mohamed Abdel-Basset, Laila Abdel-Fatah, and Arun Kumar Sangaiah. “Chapter 10 - Metaheuristic Algorithms: A Comprehensive Review”. en. In: *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*. Ed. by Arun Kumar Sangaiah, Michael Sheng, and Zhiyong Zhang. Intelligent Data-Centric Systems. Academic Press, Jan. 2018, pp. 185–231. ISBN: 978-0-12-813314-9. DOI: [10.1016/B978-0-12-813314-9.00010-4](https://doi.org/10.1016/B978-0-12-813314-9.00010-4). URL: <https://www.sciencedirect.com/science/article/pii/B9780128133149000104> (visited on 04/30/2022).
- [58] Mark Zlochin et al. “Model-Based Search for Combinatorial Optimization: A Critical Survey”. en. In: *Annals of Operations Research* 131.1 (Oct. 2004), pp. 373–395. ISSN: 1572-9338. DOI: [10.1023/B:ANOR.0000039526.52305.af](https://doi.org/10.1023/B:ANOR.0000039526.52305.af). URL: <https://doi.org/10.1023/B:ANOR.0000039526.52305.af> (visited on 04/30/2022).
- [59] Osvaldo Gomez and Benjamin Baran. “Relationship between Genetic Algorithms and Ant Colony Optimization Algorithms”. en. In: (), p. 11.
- [60] Bilal et al. “Differential Evolution: A review of more than two decades of research”. en. In: *Engineering Applications of Artificial Intelligence* 90 (Apr. 2020), p. 103479. ISSN: 0952-1976. DOI: [10.1016/j.engappai.2020.103479](https://doi.org/10.1016/j.engappai.2020.103479). URL: <https://www.sciencedirect.com/science/article/pii/S095219762030004X> (visited on 04/30/2022).
- [61] P. Larrañaga. “A Review on Estimation of Distribution Algorithms”. In: *Estimation of Distribution Algorithms*. 2002. DOI: [10.1007/978-1-4615-1539-5_3](https://doi.org/10.1007/978-1-4615-1539-5_3).
- [62] Pedro Larrañaga and Jose A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. en. Google-Books-ID: o0llxS4u93wC. Springer Science & Business Media, Oct. 2001. ISBN: 978-0-7923-7466-4.
- [63] Hydrogen Council. *Hydrogen Insights - A perspective on hydrogen investment, market development and cost competitiveness*. Feb. 2021.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl